Redfish Specification

Supersedes: 1.10.0

Document Class: Normative

Document Status: Published

Document Language: en-US
DMTF is a not-for-profit association of industry members dedicated to promoting enterprise and systems management and interoperability. Members and non-members may reproduce DMTF specifications and documents, provided that correct attribution is given. As DMTF specifications may be revised from time to time, the particular version and release date should always be noted.

Implementation of certain elements of this standard or proposed standard may be subject to third party patent rights, including provisional patent rights (herein "patent rights"). DMTF makes no representations to users of the standard as to the existence of such rights, and is not responsible to recognize, disclose, or identify any or all such third party patent right, owners or claimants, nor for any incomplete or inaccurate identification or disclosure of such rights, owners or claimants. DMTF shall have no liability to any party, in any manner or circumstance, under any legal theory whatsoever, for failure to recognize, disclose, or identify any such third party patent rights, or for such party's reliance on the standard or incorporation thereof in its product, protocols or testing procedures. DMTF shall have no liability to any party implementing such standard, whether such implementation is foreseeable or not, nor to any patent owner or claimant, and shall have no liability or responsibility for costs or losses incurred if a standard is withdrawn or modified after publication, and shall be indemnified and held harmless by any party implementing the standard from any and all claims of infringement by a patent owner for such implementations.

For information about patents held by third-parties which have notified the DMTF that, in their opinion, such patent may relate to or impact implementations of DMTF standards, visit http://www.dmtf.org/about/policies/disclosures.php.

This document's normative language is English. Translation into other languages is permitted.
CONTENTS

1. Abstract ........................................................................................................................................... 11
2. Normative references ......................................................................................................................... 11
3. Terms and definitions .......................................................................................................................... 13
4. Symbols and abbreviated terms ........................................................................................................... 16
5. Overview .......................................................................................................................................... 17
   5.1. Scope ......................................................................................................................................... 17
   5.2. Goals .......................................................................................................................................... 18
   5.3. Design tenets ............................................................................................................................... 18
   5.4. Limitations .................................................................................................................................. 19
   5.5. Additional design background and rationale .............................................................................. 19
       5.5.1. REST-based interface ........................................................................................................... 19
       5.5.2. Data-oriented ....................................................................................................................... 20
       5.5.3. Separation of protocol from data model .............................................................................. 20
       5.5.4. Hypermedia API Service Root ............................................................................................. 20
       5.5.5. OpenAPI v3.0 support ......................................................................................................... 20
       5.5.6. OData conventions .............................................................................................................. 20
   5.6. Service elements ........................................................................................................................... 21
       5.6.1. Synchronous and asynchronous operation support .............................................................. 21
       5.6.2. Eventing mechanism ............................................................................................................ 21
       5.6.3. Actions .................................................................................................................................. 21
       5.6.4. Service discovery .................................................................................................................. 21
       5.6.5. Remote access support ........................................................................................................ 22
   5.7. Security ....................................................................................................................................... 22
6. Protocol details .................................................................................................................................. 22
   6.1. Universal Resource Identifiers .................................................................................................... 23
   6.2. HTTP methods .............................................................................................................................. 25
   6.3. HTTP redirect ............................................................................................................................... 25
   6.4. Media types .................................................................................................................................. 26
   6.5. ETags ......................................................................................................................................... 26
   6.6. Protocol version ............................................................................................................................. 27
   6.7. Redfish-defined URIs and relative reference rules ................................................................. 28
7. Service requests ................................................................................................................................. 29
   7.1. Request headers ............................................................................................................................ 29
   7.2. GET (read requests) ...................................................................................................................... 35
       7.2.1. Resource collection requests .............................................................................................. 35
       7.2.2. Service Root request ........................................................................................................... 36
       7.2.3. OData service and metadata document requests .............................................................. 36
   7.3. Query parameters ......................................................................................................................... 36
       7.3.1. The $expand query parameter ............................................................................................ 39
       7.3.2. The $select query parameter ............................................................................................... 42
       7.3.3. The $filter query parameter ................................................................................................ 43
### 7.4. HEAD

- Modification requests

### 7.5. Data modification requests

- Modification success responses
- Modification error responses

### 7.6. PATCH (update)

- PATCH on array properties

### 7.7. PUT (replace)

- POST (create)
- DELETE (delete)

### 7.11. POST (Action)

- Operation apply time
- Deep operations

### 8. Service responses

#### 8.1. Response headers

- Link header

#### 8.2. Link header

- Status codes

### 8.4. OData metadata responses

#### 8.4.1. OData $metadata

- OData service document

#### 8.4.2. Message object

- Structure properties
- Array properties
- Non-resource reference properties
- Reference properties
- Date-Time values
- GUID and UUID values
- Primitive types

### 8.5. Resource responses

#### 8.6. Error responses

### 9. Data model

#### 9.1. Resources

#### 9.2. Resource collections

#### 9.3. OEM resources

#### 9.4. Common data types

##### 9.4.1. Primitive types

##### 9.4.2. GUID and UUID values

##### 9.4.3. Date-Time values

##### 9.4.4. Duration values

##### 9.4.5. Reference properties

##### 9.4.6. Non-resource reference properties

##### 9.4.7. Array properties

##### 9.4.8. Structured properties

##### 9.4.9. Message object

#### 9.5. Properties

##### 9.5.1. Resource identifier (
@odata.id
) property

##### 9.5.2. Resource type (@odata.type) property

##### 9.5.3. Resource ETag (@odata.etag) property

##### 9.5.4. Resource context (@odata.context) property

##### 9.5.5. Id

##### 9.5.6. Name

##### 9.5.7. Description
9.5.8. MemberId ................................................................. 82
9.5.9. Count (Members@odata.count) property ......................... 83
9.5.10. Members .................................................................. 83
9.5.11. Next link (Members@odata.nextLink) property .......... 83
9.5.12. Links ........................................................................ 83
9.5.13. Actions ...................................................................... 84
9.5.14. Oem .......................................................................... 86
9.5.15. Status ....................................................................... 86

9.6. Resource, schema, property, and URI naming conventions ... 86

9.7. Resource extensibility .......................................................... 88
9.7.1. OEM property format and content .................................. 88
9.7.2. OEM property naming .................................................. 89
9.7.3. OEM resource naming and URLs ................................. 90
9.7.4. OEM property examples ............................................. 90
9.7.5. OEM actions ............................................................. 91

9.8. Payload annotations ............................................................ 92
9.8.1. Allowable values ......................................................... 93
9.8.2. Extended information ................................................ 93
9.8.3. Action info annotation .............................................. 95
9.8.4. Settings and settings apply time annotations ........... 96
9.8.5. Operation apply time and operation apply time support annotations 96
9.8.6. Maintenance window annotation ................................ 96
9.8.7. Collection capabilities annotation .................................. 97
9.8.8. Requested count and allow over-provisioning annotations .... 99
9.8.9. Zone affinity annotation ........................................ 99
9.8.10. Supported certificates annotation .......................... 100
9.8.11. Deprecated annotation ........................................... 100

9.9. Settings resource ............................................................. 101

9.10. Special resource situations ............................................... 102
9.10.1. Overview .............................................................. 102
9.10.2. Absent resources .................................................... 103

9.11. Registries .................................................................... 103

9.12. Schema annotations .................................................... 104
9.12.1. Description annotation .............................................. 104
9.12.2. Long description annotation .................................... 105
9.12.3. Resource capabilities annotation .................................. 105
9.12.4. Resource URI patterns annotation .......................... 105
9.12.5. Additional properties annotation ............................. 106
9.12.6. Permissions annotation ............................................ 107
9.12.7. Required annotation .................................................. 107
9.12.8. Required on create annotation .................................... 107
9.12.9. Units of measure annotation ........................................ 107
9.12.10. Expanded resource annotation ............................ 107
9.12.11. Owning entity annotation ................................................................. 108
9.13. Versioning ............................................................................................. 108
10. File naming and publication ..................................................................... 109
   10.1. Registry file naming ........................................................................... 109
   10.2. Profile file naming ............................................................................ 110
   10.3. Dictionary file naming ...................................................................... 110
   10.4. Localized file naming ....................................................................... 110
   10.5. DMTF Redfish file repository ............................................................ 110
11. Schema definition languages .................................................................... 112
   11.1. OData Common Schema Definition Language .................................... 112
       11.1.1. File naming conventions for CSDL .............................................. 112
       11.1.2. Core CSDL files ...................................................................... 112
       11.1.3. CSDL format .......................................................................... 113
       11.1.4. Elements of CSDL namespaces .................................................... 114
   11.2. JSON Schema .................................................................................... 123
       11.2.1. File naming conventions for JSON Schema .................................. 123
       11.2.2. Core JSON Schema files ............................................................. 123
       11.2.3. JSON Schema format ................................................................ 123
       11.2.4. JSON Schema definitions body .................................................. 124
       11.2.5. JSON Schema terms ................................................................. 130
   11.3. OpenAPI ........................................................................................... 130
       11.3.1. File naming conventions for OpenAPI schema ......................... 130
       11.3.2. Core OpenAPI schema files ....................................................... 131
       11.3.3. openapi.yaml ....................................................................... 131
       11.3.4. OpenAPI file format ................................................................. 133
       11.3.5. OpenAPI components body ....................................................... 133
       11.3.6. OpenAPI terms used by Redfish ............................................... 137
   11.4. Schema modification rules ................................................................. 138
12. Service details .......................................................................................... 138
   12.1. Eventing ............................................................................................ 138
       12.1.1. POST to subscription collection .................................................... 139
       12.1.2. Open an SSE connection ........................................................... 140
       12.1.3. EventType-based eventing ......................................................... 140
       12.1.4. Subscribing to events ................................................................. 141
       12.1.5. Event formats ......................................................................... 142
       12.1.6. OEM extensions ..................................................................... 144
   12.2. Asynchronous operations ................................................................... 144
   12.3. Resource tree stability ...................................................................... 146
   12.4. Discovery .......................................................................................... 146
       12.4.1. UPnP compatibility .................................................................. 147
       12.4.2. USN format ............................................................................ 147
Foreword

The Redfish Forum of the DMTF develops the Redfish standard.

DMTF is a not-for-profit association of industry members that promotes enterprise and systems management and interoperability. For information about the DMTF, see https://www.dmtf.org/.
Acknowledgments

The DMTF acknowledges the following individuals for their contributions to the Redfish standard, including this document and Redfish schemas, interoperability profiles, and Message Registries:

- Rafiq Ahamed - Hewlett Packard Enterprise
- Richelle Ahlvers - Broadcom Inc.
- Jeff Autor - Hewlett Packard Enterprise
- David Black - Dell Inc.
- Jeff Bobzin - Insyde Software Corp.
- Patrick Boyd - Dell Inc.
- David Brockhaus - Vertiv
- Richard Brunner - VMware Inc.
- Sean Byland - Cray Inc.
- Lee Calcote - Seagate Technology
- Keith Campbell - Lenovo
- P Chandrasekhar - Dell Inc.
- Barbara Craig - Hewlett Packard Enterprise
- Chris Davenport - Hewlett Packard Enterprise
- Gamma Dean - Vertiv
- Daniel Dufresne - Dell Inc.
- Samer El-Haj-Mahmoud - Arm Limited, Lenovo, Hewlett Packard Enterprise
- George Ericson - Dell Inc.
- Wassim Fayed - Microsoft Corporation
- Kevin Ferguson - Vertiv
- Mike Garrett - Hewlett Packard Enterprise
- Steve Geffin - Vertiv
- Joe Handzik - Hewlett Packard Enterprise
- Jon Hass - Dell Inc.
- Jeff Hilland - Hewlett Packard Enterprise
- Chris Hoffman - Vertiv
- Cactus Jiang - Vertiv
- Barry Kittner - Intel Corporation
- Steven Krig - Intel Corporation
- Jennifer Lee - Intel Corporation
- John Leung - Intel Corporation
- Magnus Lundmark - Ericsson AB
- Steve Lyle - Hewlett Packard Enterprise
• Gunnar Mills - IBM
• Jagan Molleti - Dell Inc.
• Milena Natanov - Microsoft Corporation
• Scott Phuong - Cisco Systems, Inc.
• Michael Pizzo - Microsoft Corporation
• Chris Poblete - Dell Inc.
• Michael Raineri - Dell Inc.
• Irina Salvan - Microsoft Corporation
• Bill Scherer - Hewlett Packard Enterprise
• Hemal Shah - Broadcom Inc.
• Jim Shelton - Vertiv
• Tom Slaight - Intel Corporation
• Josiah Smith - Eaton
• Donnie Sturgeon - Vertiv
• Pawel Szymanski - Intel Corporation
• Paul Vancil - Dell Inc.
• Joseph White - Dell Inc.
• Linda Wu - NVIDIA Corporation, Super Micro Computer, Inc.
1. Abstract

Redfish is a standard that uses RESTful interface semantics to access a schema based data model to conduct management operations. It is suitable for a wide range of devices, from stand-alone servers, to composable infrastructures, and to large-scale cloud environments.

The initial Redfish scope targeted servers.

The DMTF and its alliance partners expanded that scope to cover most data center IT equipment and other solutions, and both in- and out-of-band access methods.

Additionally, the DMTF and other organizations that use Redfish as part of their industry standard or solution have added educational material.

2. Normative references

The following referenced documents are indispensable for the application of this document. For dated or versioned references, only the edition cited (including any corrigenda or DMTF update versions) applies. For references without a date or version, the latest published edition of the referenced document (including any corrigenda or DMTF update versions) applies.

- IETF RFC4627, D. Crockford, The application/json Media Type for JavaScript Object Notation (JSON), https://www.ietf.org/rfc/rfc4627.txt
• IETF RFC5487, M. Badra et al, Pre-Shared Key Cipher Suites for TLS with SHA-256/384 and AES Galois Counter Mode, https://www.ietf.org/rfc/rfc5487.txt
• IETF RFC6906, E. Wilde, The 'profile' Link Relation Type, https://www.ietf.org/rfc/rfc6906.txt
• IETF RFC7617, J. Reschke et al., The 'Basic' HTTP Authentication Scheme, https://www.ietf.org/rfc/rfc7617.txt
• OData Version 4.0 Plus Errata 03: Core Vocabulary. 10 March 2016. https://docs.oasis-open.org/odata/odata/v4.0/errata03/csd01/complete/vocabularies/Org.OData.Core.V1.xml
• OData Version 4.0 Plus Errata 03: Units of Measure Vocabulary. 10 March 2016. https://docs.oasis-open.org/odata/odata/v4.0/errata03/csd01/complete/vocabularies/Org.OData.Measures.V1.xml
• The OpenAPI Specification https://github.com/OAI/OpenAPI-Specification

Redfish Specification

Published

Version 1.11.0
3. Terms and definitions

Some terms and phrases in this document have specific meanings beyond their typical English meanings. This clause defines those terms and phrases.

The terms "shall" ("required"), "shall not", "should" ("recommended"), "should not" ("not recommended"), "may", "need not" ("not required"), "can" and "cannot" in this document are to be interpreted as described in ISO/IEC Directives, Part 2, Clause 7. The terms in parenthesis are alternatives for the preceding term, for use in exceptional cases when the preceding term cannot be used for linguistic reasons. Note that ISO/IEC Directives, Part 2, Clause 7 specifies additional alternatives. Occurrences of such additional alternatives shall be interpreted in their normal English meaning.

The terms "clause", "subclause", "paragraph", and "annex" in this document are to be interpreted as described in ISO/IEC Directives, Part 2, Clause 6.

The terms "normative" and "informative" in this document are to be interpreted as described in ISO/IEC Directives, Part 2, Clause 3. In this document, clauses, subclauses, or annexes labeled "(informative)" do not contain normative content. Notes and examples are always informative elements.

The term "deprecated" in this document is to be interpreted as material that is not recommended for use in new development efforts. Existing and new implementations may use this material, but they should move to the favored approach. Deprecated material may be implemented in order to achieve backwards compatibility. Deprecated material should contain references to the last published version that included the deprecated material as normative material and to a description of the favored approach. Deprecated material may be removed from the next major version of the specification.

The following typographical convention indicates deprecated material:

```
DEPRECATED

Deprecated material appears here.

END DEPRECATED
```
In places where this typographical convention cannot be used, such as tables or figures, the "DEPRECATED" label is used alone.

This document defines these additional terms:

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>baseboard management controller (BMC)</td>
<td>Embedded device or service. Typically an independent microprocessor or system-on-chip with associated firmware in a computer system that completes out-of-band systems monitoring and management-related tasks.</td>
</tr>
<tr>
<td>collection</td>
<td>See resource collection.</td>
</tr>
<tr>
<td>CRUD</td>
<td>Basic Create, Read, Update, and Delete operations that any interface can support.</td>
</tr>
<tr>
<td>event</td>
<td>Data structure that corresponds to one or more alerts.</td>
</tr>
<tr>
<td>excerpt</td>
<td>Subset of data that is copied from one resource and presented in another resource. An excerpt provides data in convenient locations without duplication of entire resources.</td>
</tr>
<tr>
<td>hypermedia API</td>
<td>API that enables you to navigate through URIs that a service returns.</td>
</tr>
<tr>
<td>managed system</td>
<td>System that provides information, status, or control through a Redfish-defined interface.</td>
</tr>
<tr>
<td>member</td>
<td>Single resource instance in a resource collection.</td>
</tr>
<tr>
<td>message</td>
<td>Complete HTTP- or HTTPS-formatted request or response. In the REST-based Redfish protocol, every request should result in a response.</td>
</tr>
<tr>
<td>OData service document</td>
<td>Resource that provides information about the Service Root for generic OData clients.</td>
</tr>
<tr>
<td>OData</td>
<td>Open Data Protocol, as defined in OData-Protocol.</td>
</tr>
<tr>
<td>operation</td>
<td>HTTP POST, GET, PUT, PATCH, HEAD, and DELETE request methods that map to generic CRUD operations.</td>
</tr>
<tr>
<td>property</td>
<td>Name-and-value pair in a Redfish-defined request or response. A property can be any valid JSON data type.</td>
</tr>
<tr>
<td>Redfish client</td>
<td>Communicates with a Redfish Service and accesses one or...</td>
</tr>
<tr>
<td>Term</td>
<td>Definition</td>
</tr>
<tr>
<td>--------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>more of the service’s resources or functions.</td>
<td></td>
</tr>
<tr>
<td>Redfish event receiver</td>
<td>Software that runs at the event destination that receives events from a Redfish Service.</td>
</tr>
<tr>
<td>Redfish protocol</td>
<td>Discovers, connects to, and inter-communicates with a Redfish Service.</td>
</tr>
<tr>
<td>Redfish provider</td>
<td>Interacts with a Redfish Service to contribute resources to the Redfish resource tree and reacts to changes in its resources. Redfish providers include:</td>
</tr>
<tr>
<td></td>
<td>• Internal provider. The Redfish Service itself that has a data model and can react to RESTful operations from a client.</td>
</tr>
<tr>
<td></td>
<td>• External provider. A designed means for agents external to the Redfish Service to augment the Redfish resource tree.</td>
</tr>
<tr>
<td></td>
<td>This specification does not describe the interaction between a Redfish provider and a Redfish Service.</td>
</tr>
<tr>
<td>Redfish schema</td>
<td>Defines Redfish resources according to OData schema representation. You can directly translate a Redfish schema to a JSON Schema representation.</td>
</tr>
<tr>
<td>Redfish Service</td>
<td>Implementation of the protocols, resources, and functions that deliver the interface that this specification defines and its associated behaviors for one or more managed systems. Also known as the service.</td>
</tr>
<tr>
<td>request</td>
<td>Message from a client to a service.</td>
</tr>
<tr>
<td>resource collection</td>
<td>Set of similar resources where the number of instances can shrink or grow.</td>
</tr>
<tr>
<td>resource tree</td>
<td>Tree structure of resources accessible through a well-known starting URI. A client may discover the available resources on a Redfish Service by following the resource hyperlinks from the base of the tree.</td>
</tr>
</tbody>
</table>
### 4. Symbols and abbreviated terms

This document uses these symbols and abbreviated terms:

<table>
<thead>
<tr>
<th>Symbol or abbreviated term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>BMC</td>
<td>Baseboard management controller</td>
</tr>
<tr>
<td>CORS</td>
<td>Cross-origin resource sharing</td>
</tr>
<tr>
<td>CRUD</td>
<td>Create, read, update, and delete</td>
</tr>
<tr>
<td>CSRF</td>
<td>Cross-Site Request Forgery</td>
</tr>
<tr>
<td>HTTP</td>
<td>Hypertext Transfer Protocol</td>
</tr>
<tr>
<td>HTTPS</td>
<td>Hypertext Transfer Protocol over TLS</td>
</tr>
<tr>
<td>IP</td>
<td>Internet Protocol</td>
</tr>
<tr>
<td>IPMI</td>
<td>Intelligent Platform Management Interface</td>
</tr>
</tbody>
</table>
5. Overview

Redfish is a management standard that uses a data model representation with a RESTful interface.

Being RESTful, Redfish is easier to use and implement.

Being model-oriented, it can express the relationships between components and the semantics of the Redfish Services and components within them. The model is also easy to extend.

By requiring JSON representation, Redfish enables easy integration with programming environments. It is also easy to interpret by humans.

An interoperable Redfish schema defines this model, which is freely available and published in OpenAPI YAML, OData CSDL, and JSON Schema formats.

5.1. Scope

This specification defines the required protocols, data model, behaviors, and other architectural components for an interoperable, multivendor, remote, and out-of-band capable interface. This interface meets the cloud-based and web-based IT professionals' expectations for scalable platform management. While large and hyperscale environments are the primary focus, clients can use the specification for individual system management.

The specification defines the required elements for all Redfish implementations, and the optional elements that system vendors and manufacturers can choose. This specification also defines at which points an implementation can provide OEM-specific extensions.

The specification sets normative requirements for Redfish Services and associated materials, such as
Redfish schema files. In general, the specification does not set requirements for Redfish clients but indicates what a client should do to successfully and effectively access and use a Redfish Service.

The specification does not require that implementations of the Redfish interfaces and functions require particular hardware or firmware.

## 5.2. Goals

As an architecture, data representation, and definition of protocols that enable a client to access Redfish Services, Redfish has these goals:

<table>
<thead>
<tr>
<th>Goal</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scalable</td>
<td>Can scale on stand-alone machines or racks of equipment.</td>
</tr>
<tr>
<td>Flexible</td>
<td>Can implement through existing hardware or entirely as a software service.</td>
</tr>
<tr>
<td>Extensible</td>
<td>Can easily add new and vendor-specific capabilities to the data model.</td>
</tr>
<tr>
<td>Backward-compatible</td>
<td>Can add capabilities while preserving investments in earlier implementations.</td>
</tr>
<tr>
<td>Interoperable</td>
<td>Provides consistent functionality across multiple vendor implementations.</td>
</tr>
<tr>
<td>Standards-based</td>
<td>Built on ubiquitous and secure protocols. Leverages other standards where applicable.</td>
</tr>
<tr>
<td>Simple</td>
<td>Easy-to-use without the need for highly specialized programming skills or systems knowledge.</td>
</tr>
<tr>
<td>Lightweight</td>
<td>Designed to reduce complexity and implementation costs. Minimizes the required footprint for implementations.</td>
</tr>
</tbody>
</table>

## 5.3. Design tenets

To deliver these goals, Redfish:

- Provides a RESTful interface by using a JSON payload and a data model.
- Separates the protocol from the data model, which enables the independent revision and use of each.
- Specifies versioning rules for protocols and schema.
- Leverages strength of ubiquitous standards where it meets architectural requirements, such as JSON, HTTP, OData, OpenAPI, and the RFCs that this document references.
- Organizes the data model so that it provides clearly demarcated and value-add features in the same payload as standardized items.
• Makes data in payloads as obvious in context as possible.
• Maintains implementation flexibility. Does not tie the interface to any particular underlying implementation or architecture.
• Focuses on widely used capabilities. To avoid complexity, does not add functions that only a small percentage of users value.

5.4. Limitations

Redfish minimizes the need for clients to complete upgrades by using strict versioning and forward-compatibility rules, and separation of the protocols from the data model. However, Redfish does not guarantee that clients never need to update their software. For example, clients might need to upgrade to manage new system or component types, or update the data model.

Interoperable does not mean identical. Many elements of Redfish are optional. Clients should be prepared to discover the optional elements by using the built-in discovery methods.

The resource tree reflects the topology of the system and its devices. Consequently, different hardware or device types result in different resource trees, even for identical systems from the same manufacturer. References between resources may result in a graph instead of a tree. Clients that traverse the resource tree should provide logic to avoid infinite loops.

Additionally, not all Redfish resources use simple REST read-and-write semantics. Different use cases may follow other types of client logic. For example, clients cannot simply read user credentials or certificates from one service and write them to another service.

Finally, the hyperlink values between resources and other elements can vary across implementations. Clients should not assume that they can reuse hyperlinks across different Redfish Service instances.

5.5. Additional design background and rationale

5.5.1. REST-based interface

Redfish exposes many service applications as RESTful interfaces. This document defines a RESTful interface.

Redfish defines a RESTful interface because it:

• Enables a lightweight implementation, using fewer layers than previous standards.
• Is a prevalent access method in the industry.
• Is easy to learn, document, and implement in modern programming languages.
• Has a number of development environments and a healthy tooling ecosystem.
• Fits with the design goal of simplicity.
• Equally applies to software application space as it does to embedded environments, which
enables convergence and sharing of code within the management ecosystem.
• Adapts well to any data modeling language.
• Has industry-provided security and discovery mechanisms.

5.5.2. Data-oriented

The Redfish data model is developed by focusing on the contents of the payload. By concentrating on the contents of the payload first, Redfish payloads are easily mapped to schema definition languages and encoding types. The data model is defined in various schema languages, including OpenAPI YAML, OData CSDL, and JSON Schema.

5.5.3. Separation of protocol from data model

Redfish separates the protocol operations from the data model and versions the protocol independently from the data model. This enables clients to extend and change the data model as needed without requiring the protocol version to change.

5.5.4. Hypermedia API Service Root

Redfish has a single Service Root URI and clients can discover all resources through referenced URIs. The hypermedia API enables the discovery of resources through hyperlinks.

5.5.5. OpenAPI v3.0 support

The OpenAPI v3.0 provides a rich ecosystem of tools for using RESTful interfaces that meet the design requirements of that specification. Starting with Redfish Specification v1.6.0, the Redfish schemas support the OpenAPI YAML file format and URI patterns that conform to the OpenAPI Specification were defined. Conforming Redfish Services that support the Redfish protocol version v1.6.0 or later implement those URI patterns to enable use of the OpenAPI ecosystem.

For details, see OpenAPI Specification v3.0.

5.5.6. OData conventions

With the popularity of RESTful APIs, there are nearly as many RESTful interfaces as there are applications. While following REST patterns helps promote good practices, due to design differences between the many RESTful APIs there few common conventions between them.

To provide for interoperability between APIs, OData defines a set of common RESTful conventions and annotations. Redfish adopts OData conventions for describing schema, URL conventions, and definitions for typical properties in a JSON payload.
5.6. Service elements

5.6.1. Synchronous and asynchronous operation support

Some operations can take more time than a client typically wants to wait. For this reason, some operations can be asynchronous at the discretion of the service. The request portion of an asynchronous operation is no different from the request portion of a synchronous operation.

To determine whether an operation was completed synchronously or asynchronously, clients can review the HTTP status codes. For more information, see the Asynchronous operations clause.

5.6.2. Eventing mechanism

Redfish enables clients to receive messages outside the normal request and response paradigm. The service uses these messages, or events, to asynchronously notify the client of a state change or error condition, usually of a time critical nature.

This specification defines two styles of eventing:

- Push-style eventing.
  
  When the service detects the need to send an event, it calls HTTP POST to push the event message to the client. Clients can enable reception of events by creating a subscription entry in the Event Service, or an administrator can create subscriptions as part of the Redfish Service configuration.

- Server-Sent Events (SSE)-style eventing.
  
  The client opens an SSE connection to the service through a GET on the ServerSentEventUri-specified URI in the Event Service.

For information, see the Eventing clause.

5.6.3. Actions

Actions are Redfish operations that do not easily map to RESTful interface semantics. These types of operations may not directly affect properties in the Redfish resources. The Redfish schema defines certain standard actions for common Redfish resources. For these standard actions, the Redfish schema contains the normative language on the behavior of the action.

5.6.4. Service discovery

While the service itself is at a well-known URI, clients need to discover the network address of the service. Like UPnP, Redfish uses SSDP for discovery. A wide variety of devices, such as printers and
client operating systems, support SSDP. It is simple, lightweight, IPv6 capable, and suitable for implementation in embedded environments.

For more information, see the Discovery clause.

### 5.6.5. Remote access support

Remote management functionality typically includes access mechanisms for redirecting operator interfaces such as serial console, keyboard video and mouse (KVM-IP), command shell, or command-line interface, and virtual media. While these mechanisms are critical functionality, they cannot be reasonably implemented as a RESTful interface.

Therefore, this standard does not define the protocols or access mechanisms for those services but encourages implementations that leverage existing standards. However, the Redfish schema includes resources and properties that enable client discovery of these capabilities and access mechanisms to enable interoperability.

### 5.7. Security

The challenge of remote interface security is to protect both the interface and exchanged data. To accomplish this, Redfish provides authentication and encryption. As part of this security, Redfish defines and requires minimum levels of encryption.

For more information, see the Security details clause.

### 6. Protocol details

In this document, the Redfish protocol refers to the RESTful mapping to HTTP, TCP/IP, and other protocol, transport, and messaging layer aspects. HTTP is the application protocol that transports the messages and TCP/IP is the transport protocol. The RESTful interface is a mapping to the message protocol.

The Redfish protocol is designed around a web service-based interface model. This provides network and interaction efficiency for both user interface (UI) and automation usage. Specifically, the protocol can leverage existing tool chains.

The Redfish protocol uses these items for these purposes:

<table>
<thead>
<tr>
<th>Item</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>HTTP methods</td>
<td>Maps to common CRUD operations.</td>
</tr>
<tr>
<td>Actions</td>
<td>Expands operations beyond CRUD-type operations.</td>
</tr>
</tbody>
</table>
### Item

<table>
<thead>
<tr>
<th>Item</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>Media types</td>
<td>Negotiates the type of data sent in the message body.</td>
</tr>
<tr>
<td>HTTP status codes</td>
<td>Indicates the success or failure of the server's request.</td>
</tr>
<tr>
<td>Error responses</td>
<td>Returns more information than HTTP status codes.</td>
</tr>
<tr>
<td>TLS</td>
<td>Secures messages. See Security details.</td>
</tr>
<tr>
<td>Asynchronous semantics</td>
<td>For long-running operations.</td>
</tr>
</tbody>
</table>


The subsequent clauses describe how the Redfish interface uses and adds constraints to HTTP to ensure interoperability of Redfish implementations.

### 6.1. Universal Resource Identifiers

A Universal Resource Identifier (URI) identifies a resource, including the Service Root and all Redfish resources.

- A URI shall identify each unique instance of a resource.
- URIs shall not include any [RFC1738](https://tools.ietf.org/html/rfc1738)-defined unsafe characters.
  - For example, the ``, ``, ``, ``, ``, ``, and `\` characters are unsafe because gateways and other transport agents can sometimes modify these characters.
  - Do not use the `#` character for anything other than the start of a fragment.
- URIs shall not include any percent-encoding of characters. This restriction does not apply to the **query parameters** portion of the URI.

Performing a GET operation on a URI returns a representation of the resource with properties and hyperlinks to associated resources. The Service Root URI is well known and is based on the protocol version.

To discover the URIs to additional resources, extract the associated resource hyperlinks from earlier responses. The **hypermedia API** enables the discovery of resources through hyperlinks.

Redfish considers the [RFC3986](https://tools.ietf.org/html/rfc3986)-defined scheme, authority, Service Root, and version, and unique resource path component parts of the URI.

For example, this URI:
Contains these component parts:

<table>
<thead>
<tr>
<th>Component part</th>
<th>Defines</th>
</tr>
</thead>
<tbody>
<tr>
<td>https:</td>
<td>Scheme.</td>
</tr>
<tr>
<td>mgmt.vendor.com</td>
<td>Authority to which to delegate the URI.</td>
</tr>
<tr>
<td>redfish/v1</td>
<td>Service Root and version.</td>
</tr>
<tr>
<td>Systems/1</td>
<td>Unique resource path.</td>
</tr>
</tbody>
</table>

In a URI:

- The scheme and authority component parts are not part of the unique resource path because redirection capabilities and local operations may cause the connection portion to vary.
- The Service Root and resource path component parts uniquely identify the resource in a Redfish Service.

In an implementation:

- The resource path component part shall be unique.
- A relative reference in the body and HTTP headers payload can identify a resource in that same implementation.
- An absolute URI in the body and HTTP headers payload can identify a resource in a different implementation.

For the absolute URI definition, see RFC3986.

For example, a POST operation may return the `/redfish/v1/Systems/2` URI in the Location header of the response, which points to the POST-created resource.

Assuming that the client connects through the mgmt.vendor.com appliance, the client accesses the resource through the https://mgmt.vendor.com/redfish/v1/Systems/2 absolute URI.

URIs that conform to RFC3986 may also contain the query, `?query`, and frag, `#frag`, components. For information about queries, see Query parameters. When a URI includes a fragment (`frag`) to submit an operation, the server ignores the fragment.

If a property in a response references another property within a resource, use the RFC6901-defined URI fragment identifier representation format. If the property is a reference property in the schema, the fragment shall reference a valid resource identifier. For example, the following fragment identifies a
property at index 0 of the Fans array in the /redfish/v1/Chassis/MultiBladeEncl/Thermal resource:

```json
{
    "@odata.id": "'/redfish/v1/Chassis/MultiBladeEncl/Thermal#/Fans/0"
}
```

For requirements on constructing Redfish URIs, see the resource URI patterns annotation clause.

### 6.2. HTTP methods

The following table describes the mapping of HTTP methods to the operations that are supported by Redfish. The Required column specifies whether a Redfish interface supports the method.

- If the value is Yes, the HTTP method shall be supported.
- If the value is No, the value may be supported.

For HTTP methods that the Redfish service does not support or that the following table omits, the Redfish service shall return the HTTP 405 Method Not Allowed status code or the HTTP 501 Not Implemented status code.

<table>
<thead>
<tr>
<th>HTTP method</th>
<th>Interface semantic</th>
<th>Required</th>
</tr>
</thead>
<tbody>
<tr>
<td>POST</td>
<td>Resource create</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>Resource action</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Eventing</td>
<td></td>
</tr>
<tr>
<td>GET</td>
<td>Resource retrieval</td>
<td>Yes</td>
</tr>
<tr>
<td>PUT</td>
<td>Resource replace</td>
<td>No</td>
</tr>
<tr>
<td>PATCH</td>
<td>Resource update</td>
<td>Yes</td>
</tr>
<tr>
<td>DELETE</td>
<td>Resource delete</td>
<td>Yes</td>
</tr>
<tr>
<td>HEAD</td>
<td>Resource header retrieval</td>
<td>No</td>
</tr>
<tr>
<td>OPTIONS</td>
<td>Header retrieval</td>
<td>No</td>
</tr>
<tr>
<td></td>
<td>Cross-origin resource sharing (CORS) preflight</td>
<td></td>
</tr>
</tbody>
</table>

### 6.3. HTTP redirect

HTTP redirect enables a service to redirect a request to another URL. Among other things, HTTP redirect
enables Redfish resources to alias areas of the data model.

- All Redfish clients shall correctly handle HTTP redirect.

**Note:** For security implications of HTTP redirect, see [Security details](#).

### 6.4. Media types

Some resources may be available in more than one type of representation. The media type indicates the representation type.

In HTTP messages, the media type is specified in the `Content-Type` header. To tell a service to send the response through certain media types, the client sets the HTTP `Accept` header to a list of the media types.

- All resources shall be available through the JSON `application/json` media type.
- Redfish Services shall make every resource available in a JSON-based representation, as specified in [RFC4627](https://tools.ietf.org/html/rfc4627). Receivers shall not reject a JSON-encoded message, and shall offer at least one JSON-based response representation. An implementation may offer additional non-JSON media type representations.

To request compression, clients specify an `Accept-Encoding` request header.

When requested by the client, services should support gzip compression.

### 6.5. ETags

To reduce unnecessary RESTful accesses to resources, the Redfish Service should support the association of a separate entity tag (ETag) with each resource.

- Implementations should support the return of ETag properties for each resource.
- Implementations should support the return of ETag headers for each single-resource response.
- Implementations shall support the return of ETag headers for GET requests of ManagerAccount resources.

Because the service knows whether the new version of the object is substantially different, the service generates and provides the ETag as part of the resource payload.

The ETag mechanism supports both **strong** and **weak** validation. If a resource supports an ETag, it shall use the [RFC7232](https://tools.ietf.org/html/rfc7232)-defined ETag strong validator.

This specification does not mandate a particular algorithm for ETag creation, but ETags should be highly collision-free.
An ETag can be:

- A hash
- A generation ID
- A time stamp
- Some other value that changes when the underlying object changes

If a client calls PUT or PATCH to update a resource, it should include an ETag from a previous GET in the HTTP If-Match or If-None-Match header. If a service supports the return of the ETag header on a resource, it may respond with the HTTP 428 Precondition Required status code if the If-Match or If-None-Match header is missing from the PUT or PATCH request for the same resource, as specified in RFC6585.

In addition to the return of the ETag property on each resource, a Redfish Service should return the ETag header on:

- A client PUT, POST, or PATCH operation
- A GET operation for an individual resource

The format of the ETag header is:

```
ETag: "<string>"
```

### 6.6. Protocol version

The protocol version is separate from the resources' version or the Redfish schema version that the resources support.

Each Redfish protocol version is strongly typed by using the URI of the Redfish Service in combination with the resource obtained at that URI, called the ServiceRoot resource.

The root URI for this version of the Redfish protocol shall be /redfish/v1/.

The URI defines the major version of the protocol.

The RedfishVersion property of the ServiceRoot resource defines the protocol version, which includes the major version, minor version, and errata version of the protocol, as defined in the Redfish schema for that resource.

The protocol version is a string in the format:

```
```
where

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
<th>Version</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MajorVersion</td>
<td>Integer</td>
<td>Major</td>
<td>Backward-compatible class change.</td>
</tr>
<tr>
<td>MinorVersion</td>
<td>Integer</td>
<td>Minor</td>
<td>Minor update. Redfish introduces functionality but does not remove any functionality. The minor version preserves compatibility with earlier minor versions.</td>
</tr>
<tr>
<td>ErrataVersion</td>
<td>Integer</td>
<td>Errata</td>
<td>Fix to the earlier version.</td>
</tr>
</tbody>
</table>

Any resource that a client discovers through hyperlinks that the Service Root or any Service Root-referenced service or resource returns shall conform to the same protocol version that the Service Root supports.

A GET operation on the /redfish resource shall return this response body:

```json
{
  "v1": "/redfish/v1/
}
```

6.7. Redfish-defined URIs and relative reference rules

A Redfish Service shall support these Redfish-defined URIs:

<table>
<thead>
<tr>
<th>URI</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>/redfish</td>
<td>The version. A major update that does not preserve compatibility with earlier minor versions.</td>
</tr>
<tr>
<td>/redfish/v1/</td>
<td>The Redfish Service Root.</td>
</tr>
<tr>
<td>/redfish/v1/odata</td>
<td>The Redfish OData service document.</td>
</tr>
<tr>
<td>/redfish/v1/$metadata</td>
<td>The Redfish metadata document.</td>
</tr>
</tbody>
</table>
A Redfish Service should support these Redfish-defined URIs:

<table>
<thead>
<tr>
<th>URI</th>
<th>Returns</th>
</tr>
</thead>
<tbody>
<tr>
<td>/redfish/v1/openapi.yaml</td>
<td>The Redfish OpenAPI YAML document</td>
</tr>
</tbody>
</table>

In addition, the service shall process the following URI without a trailing slash in one of these ways:

- Redirect it to the associated Redfish-defined URI.
- Treat it as the equivalent URI to the associated Redfish-defined URI:

<table>
<thead>
<tr>
<th>URI</th>
<th>Associated Redfish-defined URI</th>
</tr>
</thead>
<tbody>
<tr>
<td>/redfish/v1</td>
<td>/redfish/v1/</td>
</tr>
<tr>
<td>/redfish</td>
<td>/redfish</td>
</tr>
</tbody>
</table>

All other Redfish Service-supported URIs shall match the resource URI patterns definitions, except the supplemental resources that the @Redfish.Settings, @Redfish.ActionInfo, and @Redfish.CollectionCapabilities payload annotations reference. The client shall treat the URIs for these supplemental resources as opaque.

All Redfish Service-supported URIs are reserved for future standardization by DMTF and DMTF alliance partners, except OEM extension URIs, which shall conform to the OEM resource URI requirements.

All relative references that the service uses shall start with either:

- A double forward slash (//) and include the authority (network-path), such as //mgmt.vendor.com/redfish/v1/Systems.
- A single forward slash (/) and include the absolute-path, such as /redfish/v1/Systems.

For details, see RFC3986.

7. Service requests

This clause describes the requests that clients can send to Redfish Services.

7.1. Request headers

The HTTP Specification defines headers for request messages. The following table defines those headers and their requirements for Redfish Services and clients.

For Redfish Services:
• Redfish Services shall process the headers in the following table as defined by the HTTP 1.1 Specification if the value in the **Service requirement** column is **Yes**, or if the value is **Conditional** under the conditions noted in the Description column.

• Redfish Services should process the headers in the following tables as defined by the HTTP 1.1 Specification if the value in the **Service requirement** column is **No**.

For Redfish clients (sending the HTTP requests):

• Redfish clients shall include the headers in the following table as defined by the HTTP 1.1 Specification if the value in the **Client requirement** column is **Yes**, or if the value is **Conditional** under the conditions noted in the Description column.

• Redfish clients should transmit the headers in the following tables as defined by the HTTP 1.1 Specification if the value in the **Client requirement** column is **No**.

<table>
<thead>
<tr>
<th>Header</th>
<th>Service requirement</th>
<th>Client requirement</th>
<th>Supported values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accept</td>
<td>Yes</td>
<td>No</td>
<td>RFC7231</td>
<td>Communicates to the server the media type or types that this client is prepared to accept. Services shall support resource requests with Accept header values of application/json or application/json; charset=utf-8. Services shall support XML metadata requests with Accept header values of application/xml or application/xml; charset=utf-8. Services shall support OpenAPI YAML schema requests with Accept header values of application/yaml or application/yaml; charset=utf-8 or application/vnd.oai.openapi or application/vnd.oai.openapi; charset=utf-8. Services shall support SSE requests</td>
</tr>
<tr>
<td>Header</td>
<td>Service requirement</td>
<td>Client requirement</td>
<td>Supported values</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>---------------------</td>
<td>--------------------</td>
<td>------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Accept-Encoding</td>
<td>No</td>
<td>No</td>
<td>RFC7231</td>
<td>Indicates whether the client can handle gzip-encoded responses.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>If a service cannot send an acceptable response to a request with this header, it shall respond with the HTTP 406 Not Acceptable status code.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>If the request omits this header, the service shall not return gzip-encoded responses.</td>
</tr>
<tr>
<td>Accept-Language</td>
<td>No</td>
<td>No</td>
<td>RFC7231</td>
<td>The languages that the client accepts in the response.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>If the request omits this header, uses the service’s default language for the response.</td>
</tr>
<tr>
<td>Authorization</td>
<td>Conditional</td>
<td>Conditional</td>
<td>RFC7617</td>
<td>Required for HTTP basic authentication.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>A client can access unsecured resources without this header on systems that support basic authentication.</td>
</tr>
</tbody>
</table>

With `Accept` header values of `text/event-stream` or `text/event-stream; charset=utf-8`. Services shall support any request with `Accept` header values of `application/*`, `application/*; charset=utf-8`, `*/*`, or `*/*; charset=utf-8`.
<table>
<thead>
<tr>
<th>Header</th>
<th>Service requirement</th>
<th>Client requirement</th>
<th>Supported values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Content-Length</td>
<td>No</td>
<td>No</td>
<td>RFC7231</td>
<td>The size of the message body. To indicate the size of the body, a client can use the Transfer-Encoding: chunked header. If a service needs to use Content-Length and does not support Transfer-Encoding, it responds with the HTTP 406 Not Acceptable status code.</td>
</tr>
<tr>
<td>Content-Type</td>
<td>Conditional</td>
<td>Conditional</td>
<td>RFC7231</td>
<td>The request format. Required for operations with a request body. Services shall accept the Content-Type header set to either application/json or application/json; charset=utf-8. It is recommended that clients use these values in requests because other values can cause an error.</td>
</tr>
<tr>
<td>Host</td>
<td>Yes</td>
<td>No</td>
<td>RFC7230</td>
<td>Enables support of multiple origin hosts at a single IP address.</td>
</tr>
<tr>
<td>If-Match</td>
<td>Conditional</td>
<td>No</td>
<td>RFC7232</td>
<td>To ensure that clients update the resource from a known state, PUT and PATCH requests for resources for</td>
</tr>
<tr>
<td>Header</td>
<td>Service requirement</td>
<td>Client requirement</td>
<td>Supported values</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------------</td>
<td>---------------------</td>
<td>--------------------</td>
<td>-----------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>If-None-Match</td>
<td>No</td>
<td>No</td>
<td>RFC7232</td>
<td>A service only returns the resource if the current ETag of that resource does not match the ETag sent in this header. If the ETag in this header matches the resource's current ETag, the GET operation returns the HTTP 304 Not Modified status code.</td>
</tr>
<tr>
<td>Last-Event-ID</td>
<td>No</td>
<td>No</td>
<td>HTML5 SSE</td>
<td>The event source's last id field from the SSE stream. Requests history event data. See Server-Sent Events.</td>
</tr>
<tr>
<td>Max-Forwards</td>
<td>No</td>
<td>No</td>
<td>RFC7231</td>
<td>Limits gateway and proxy hops. Prevents messages from remaining in the network indefinitely.</td>
</tr>
<tr>
<td>OData-MaxVersion</td>
<td>No</td>
<td>No</td>
<td>4.0</td>
<td>The maximum OData version that an OData-aware client understands.</td>
</tr>
<tr>
<td>OData-Version</td>
<td>Yes</td>
<td>No</td>
<td>4.0</td>
<td>The OData version.</td>
</tr>
</tbody>
</table>
Redfish Services shall understand and be able to process the headers in the following table as defined by this specification if the value in the Required column is Yes.

<table>
<thead>
<tr>
<th>Header</th>
<th>Service requirement</th>
<th>Client requirement</th>
<th>Supported values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>X-Auth-Token</td>
<td>Yes</td>
<td>Conditional</td>
<td>Opaque encoded octet strings</td>
<td>Authenticates user sessions. The token value shall be indistinguishable from random. While services shall support this header, a client can access unsecured resources without</td>
</tr>
</tbody>
</table>

Services shall reject requests that specify an unsupported OData version.

If a service encounters an unsupported OData version, it should reject the request with the HTTP [412 Precondition Failed] status code.

**Origin**

| Yes             | No                  | W3C CORS, Section 5.7 | Enables web applications to consume a Redfish Service while preventing CSRF attacks. |

**User-Agent**

| Yes             | No                  | RFC7231               | Traces product tokens and their versions. The header can list multiple product tokens. |

**Via**

| No              | No                  | RFC7230               | Defines the network hierarchy and recognizes message loops. Each pass inserts its own Via header. |
7.2. GET (read requests)

The GET operation retrieves resources from a Redfish Service. Clients make a GET request to the individual resource URI. Clients may obtain the resource URI from published sources, such as the OpenAPI document, or from a resource identifier property in a previously retrieved resource response, such as the links property.

The service shall return the resource representation using one of the media types listed in the Accept header, subject to the requirements of the media types. If the Accept header is absent, the service shall return the resource's representation as application/json. Services may, but are not required to, support the convention of retrieving individual properties within a resource by appending a segment containing the property name to the URI of the resource.

- The HTTP GET operation shall retrieve a resource without causing any side effects.
- The service shall ignore the content of the body on a GET.
- The GET operation shall be idempotent in the absence of outside changes to the resource.

7.2.1. Resource collection requests

Clients retrieve a resource collection by making a GET request to the resource collection URI. The response includes the resource collection's properties and an array of its members.

No requirements are placed on implementations to return a consistent set of members when a series of requests that use paging query parameters are made over time to obtain the entire set of members. These calls can result in missed or duplicate elements if multiple GETs use paging to retrieve the Members array instances.

- Clients shall not make assumptions about the URIs for the members of a resource collection.
- Retrieved resource collections shall always include the count property to specify the total number of entries in its Members array.
- Regardless of the next link property or paging, the count property shall return the total number of resources that the Members array references.

A subset of the members can be retrieved using client paging query parameters.

A service may not be able to return all of the contents of a resource collection request in a single response body. In this case, the response can be paged by the service. If a service pages a response to a resource collection request, the following rules shall apply:

<table>
<thead>
<tr>
<th>Header</th>
<th>Service requirement</th>
<th>Client requirement</th>
<th>Supported values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>establishing a session.</td>
</tr>
</tbody>
</table>
• Responses may contain a subset of the full resource collection's members.
• Individual members shall not be split across response bodies.
• A next link annotation shall be supplied in the response body with the URI to the next set of members in the collection.
• The next link property shall adhere to the rules in the Next link property clause.
• GET Operations on the next link shall return the subsequent section of the resource collection response.

7.2.2. Service Root request

The root URL for Redfish version 1.x services shall be /redfish/v1/.

The service returns the ServiceRoot resource, as defined by this specification, as a response for the root URL.

Services shall not require authentication to retrieve the Service Root and /redfish resources.

7.2.3. OData service and metadata document requests

Redfish Services expose two OData-defined documents at specific URIs to enable generic OData clients to navigate the Redfish Service.

• Service shall expose an OData metadata document at the /redfish/v1/$metadata URI.
• Service shall expose an OData service document at the /redfish/v1/odata URI.
• Service shall not require authentication to retrieve the OData metadata document or the OData service document.

7.3. Query parameters

To paginate, retrieve subsets of resources, or expand the results in a single response, clients can include the query parameters. Some query parameters apply only to resource collections.

Services:

• Shall only support query parameters on GET operations.
• Should support the $top, $skip, only, and excerpt query parameters.
• May support the $expand, $filter, and $select query parameters.
• Shall include the ProtocolFeaturesSupported object in the Service Root, if the service supports query parameters.
  ◦ This object indicates which parameters and options have been implemented.
• Shall ignore unknown or unsupported query parameters that do not begin with $.
• Shall use the & operator to separate multiple query parameters in a single request.
Services shall return:

- The HTTP 501 Not Implemented status code for any unsupported query parameters that start with $.
- An extended error that indicates the unsupported query parameters for this resource.
- The HTTP 400 Bad Request status code for any query parameters that contain values that are invalid, or values applied to query parameters without defined values, such as excerpt or only.

Services should return:

- The HTTP 400 Bad Request status code with the QueryNotSupportedOnResource message from the Base Message Registry for any implemented query parameters that are not supported on a resource in the request.
- The HTTP 400 Bad Request status code with the QueryNotSupportedOnResource message from the Base Message Registry for any supported query parameters that apply only to resource collections but are used on singular resources. This includes query parameters such as $filter, $top, $skip, and only.
- The HTTP 400 Bad Request status code with the QueryNotSupportedOnOperation message from the Base Message Registry for any supported query parameters on operations other than GET.

The response body shall reflect the evaluation of the query parameters in this order:

- Prior to service-side pagination: $filter, $skip, $top
- After applying any service-side pagination: $expand, $select

<table>
<thead>
<tr>
<th>Query parameter</th>
<th>Description and example</th>
</tr>
</thead>
</table>
| excerpt         | Returns a subset of the resource's properties that match the defined Excerpt schema annotation.  
If no Excerpt schema annotation is defined for the resource, the entire resource is returned.  
**Example:**  
https://resource?excerpt |
<p>| $expand=&lt;string&gt; | Returns a hyperlink and its contents in-line with retrieved resources, as if a GET call response was included in-line with that hyperlink. |</p>
<table>
<thead>
<tr>
<th>Query parameter</th>
<th>Description and example</th>
</tr>
</thead>
</table>
| $expand=*$($levels=3) | See [The $expand query parameter](#).  
Example:  
https://resource?$expand=*$($levels=3)  
https://resourcecollection?$expand=.*($levels=1) |
| $filter=<string> | Applies to resource collections. Returns a subset of collection members that match the $filter expression.  
See [The $filter query parameter](#).  
Example:  
https://resourcecollection?$filter=SystemType eq 'Physical' |
| only | Applies to resource collections. If the target resource collection contains exactly one member, clients can use this query parameter to return that member's resource.  
If the collection contains either zero members or more than one member, the response returns the resource collection, as expected.  
Services should return the HTTP 400 Bad Request with the QueryCombinationInvalid message from the Base Message Registry if only is being combined with other query parameters.  
Example:  
https://resourcecollection?only |
| $select=<string> | Returns a subset of the resource's properties that match the $select expression.  
See [The $select query parameter](#). |
### Query parameter | Description and example
--- | ---
| Example: | https://resource?$select=SystemType,Status |

#### $skip=<integer>
Applies to resource collections. Returns a subset of the members in a resource collection. This paging query parameter defines the number of members in the resource collection to skip.

**Example:**

https://resourcecollection?$skip=5

#### $top=<integer>
Applies to resource collections. Defines the number of members to show in the response.

Minimum value is 1. By default, returns all members.

**Example:**

https://resourcecollection?$top=30

### 7.3.1. The $expand query parameter

The $expand query parameter enables a client to request a response that includes not only the requested resource, but also includes the contents of the subordinate or hyperlinked resources. The definition of this query parameter follows the OData-Protocol Specification.

The $expand query parameter has a set of possible options that determine which hyperlinks in a resource are included in the expanded response. Some resources may already be expanded due to the resource's schema annotation AutoExpand, such as the Temperature object in the Thermal resource.

The Redfish-supported options for the $expand query parameter are listed in the following table. The service may implement some of these options but not others. Any other supported syntax for $expand is outside the scope of this specification.
<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>asterisk (*)</td>
<td>Shall expand all hyperlinks, including those in payload annotations, such as @Redfish.Settings, @Redfish.ActionInfo, and @Redfish.CollectionCapabilities.</td>
<td><a href="https://resource?$expand=">https://resource?$expand=</a>*</td>
</tr>
<tr>
<td>$levels</td>
<td>Number of levels the service should cascade the $expand operation. The default level shall be 1. For example, $levels=2 expands both: • The hyperlinks in the current resource (level 1). • The hyperlinks in the resulting expanded resources (level 2).</td>
<td><a href="https://resourcecollection?$expand=.($levels=2)">https://resourcecollection?$expand=.($levels=2)</a></td>
</tr>
<tr>
<td>period (.)</td>
<td>Shall expand all hyperlinks not in any links property instances of the resource, including those in payload annotations, such as @Redfish.Settings, @Redfish.ActionInfo, and @Redfish.CollectionCapabilities.</td>
<td><a href="https://resourcecollection?$expand=">https://resourcecollection?$expand=</a>.</td>
</tr>
<tr>
<td>tilde (~)</td>
<td>Shall expand all hyperlinks found in all links property instances of the resource.</td>
<td><a href="https://resourcecollection?$expand=">https://resourcecollection?$expand=</a>~</td>
</tr>
</tbody>
</table>

Examples of $expand usage include:

- **GET of a SoftwareInventoryCollection.**

  With $expand, the client can request multiple SoftwareInventory collection member resources in one request rather than fetching them one at a time.

- **GET of a ComputerSystem.**

  With $levels, a single GET request can include the subordinate resource collections, such as Processors and Memory.

- **GET all UUIDs in members of the ComputerSystem collection.**
To accomplish this result, include both $select and $expand on the URI.

The syntax is GET `/redfish/v1/Systems?$select=UUID&$expand=.($levels=1)`

When services execute $expand, they may omit some of the referenced resource's properties.

When clients use $expand, they should be aware that the payload may increase beyond what can be sent in a single response.

If a service cannot return the payload due to its size, it shall return the HTTP 507 Insufficient Storage status code.

If a service cannot return the payload corresponding to an individual member of a resource collection, it should return the @odata.id property for that member and should return extended information indicating the reason that member was not returned, such as when a provider internal to the service returns an error or times out.

The following is an example showing the RoleCollection resource being expanded with the level set to 1:

```json
{
    "@odata.id": "/redfish/v1/AccountService/Roles",
    "@odata.type": "#RoleCollection.RoleCollection",
    "Name": "Roles Collection",
    "Members@odata.count": 3,
    "Members": [
        {
            "@odata.id": "/redfish/v1/AccountService/Roles/Administrator",
            "@odata.type": "#Role.v1_1_0.Role",
            "Id": "Administrator",
            "Name": "User Role",
            "Description": "Admin User Role",
            "IsPredefined": true,
            "AssignedPrivileges": [
                "Login",
                "ConfigureManager",
                "ConfigureUsers",
                "ConfigureSelf",
                "ConfigureComponents"
            ]
        },
        {
            "@odata.id": "/redfish/v1/AccountService/Roles/Operator",
            "@odata.type": "#Role.v1_1_0.Role",
            "Id": "Operator",
            "Name": "User Role",
```
7.3.2. The $select query parameter

The $select query parameter indicates that the implementation should return a subset of the resource's properties that match the $select expression. If a request omits the $select query parameter, the response returns all properties by default. The definition of this query parameter follows the OData-Protocol Specification.

The $select expression shall not affect the resource itself.

The $select expression defines a comma-separated list of properties to return in the response body.

The syntax for properties in object types shall be the object and property names concatenated with a slash (/).

An example of $select usage is:

```
GET /redfish/v1/Systems/1?$select=Name,SystemType,Status/State
```

When services execute $select, they shall return all requested properties of the referenced resource. The @odata.id and @odata.type properties shall be in the response payload and contain the same
values as if $select was not performed. If the `$odata.context` property is supported, it shall be in the response payload and should be in the `Context property` recommended format. If the `$odata.etag` property is supported, it shall be in the response payload and contain the same values as if $select was not performed.

Any other supported syntax for $select is outside the scope of this specification.

### 7.3.3. The $filter query parameter

The `$filter` parameter enables a client to request a subset of the resource collection's members based on the `$filter` expression. The definition of this query parameter follows the [OData-Protocol Specification](#).

The `$filter` query parameter defines a set of properties and literals with an operator. A literal value can be:

- A string enclosed in single quotes.
- A number.
- A boolean value.

If the literal value does not match the data type for the specified property, the service should reject `$filter` requests with the HTTP 400 Bad Request status code.

The `$filter` section of the OData ABNF Components Specification contains the grammar for the allowable syntax of the `$filter` query parameter, with the additional restriction that only built-in filter operations are supported.

The following table lists the Redfish-supported values for the `$filter` query parameter. Any other supported syntax for `$filter` is outside the scope of this specification.

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>()</td>
<td>Precedence grouping operator.</td>
<td>(Status/State eq 'Enabled' and Status/Health eq 'OK') or SystemType eq 'Physical'</td>
</tr>
<tr>
<td>and</td>
<td>Logical and operator.</td>
<td>ProcessorSummary/Count eq 2 and MemorySummary/TotalSystemMemoryGiB gt 64</td>
</tr>
<tr>
<td>eq</td>
<td>Equal comparison operator.</td>
<td>ProcessorSummary/Count eq 2</td>
</tr>
<tr>
<td>ge</td>
<td>Greater than or equal to comparison operator.</td>
<td>ProcessorSummary/Count ge 2</td>
</tr>
<tr>
<td>Value</td>
<td>Description</td>
<td>Example</td>
</tr>
<tr>
<td>-------</td>
<td>--------------------------------------</td>
<td>----------------------------------------------</td>
</tr>
<tr>
<td>gt</td>
<td>Great than comparison operator.</td>
<td>ProcessorSummary/Count gt 2</td>
</tr>
<tr>
<td>le</td>
<td>Less than or equal to comparison operator.</td>
<td>MemorySummary/TotalSystemMemoryGiB le 64</td>
</tr>
<tr>
<td>lt</td>
<td>Less than comparison operator.</td>
<td>MemorySummary/TotalSystemMemoryGiB lt 64</td>
</tr>
<tr>
<td>ne</td>
<td>Not equal comparison operator.</td>
<td>SystemType ne 'Physical'</td>
</tr>
<tr>
<td>not</td>
<td>Logical negation operator.</td>
<td>not (ProcessorSummary/Count eq 2)</td>
</tr>
<tr>
<td>or</td>
<td>Logical or operator.</td>
<td>ProcessorSummary/Count eq 2 or ProcessorSummary/Count eq 4</td>
</tr>
</tbody>
</table>

When evaluating expressions, services shall use the following operator precedence:

- Grouping
- Logical negation
- Relational comparison. `gt`, `ge`, `lt`, and `le` all have equal precedence.
- Equality comparison. `eq` and `ne` both have equal precedence.
- Logical `and`
- Logical `or`

If the service receives an unsupported `$filter` query parameter, it shall reject the request and return the HTTP 501 Not Implemented status code.

### 7.4. HEAD

The HEAD method differs from the GET method in that it shall not return message body information.

However, the HEAD method completes the same authorization checks and returns all the same meta information and status codes in the HTTP headers as a GET method.

Services may support the HEAD method to:

- Return meta information in the form of HTTP response headers.
- Verify hyperlink validity.

Services may support the HEAD method to verify resource accessibility.
Services shall not support any other use of the HEAD method.

The HEAD method shall be idempotent in the absence of outside changes to the resource.

7.5. Data modification requests

To create, modify, and delete resources, clients issue the following operations:

- **POST (create)**
- **PATCH (update)**
- **PUT (replace)**
- **DELETE (delete)**
- **POST (action) on the resource**

The following clauses describe the success and error response requirements common to all data modification requests.

7.5.1. Modification success responses

For create operations, the response from the service, after the create request succeeds, should be one of these responses:

- The HTTP **201 Created** status code with a body that contains the JSON representation of the newly created resource after the request has been applied.
- The HTTP **202 Accepted** status code with a **Location** header set to the URI of a task monitor when the processing of the request requires additional time to be completed.
  - After processing of the task is complete, the created resource may be returned in response to a request to the task monitor URI with the HTTP **201 Created** status code.
- The HTTP **204 No Content** status code with empty payload in the event that the service cannot return a representation of the created resource.

For update, replace, and delete operations, the response from the service, after successful modification, should be one of the following responses:

- The HTTP **200 OK** status code with a body that contains the JSON representation of the targeted resource after the modification has been applied, or, for the delete operation, a representation of the deleted resource.
- The HTTP **202 Accepted** status code with a **Location** header set to the URI of a task monitor when the processing of the modification requires additional time.
  - After processing of the task is complete, the modified resource may be returned in response to a request to the task monitor URI with the HTTP **200 OK** status code.
- The HTTP **204 No Content** status code with an empty payload in the event that service
cannot return a representation of the modified or deleted resource.

For details on successful responses to action requests, see POST (action).

### 7.5.2. Modification error responses

If the resource exists but does not support the requested operation, services may return the HTTP `405 Method Not Allowed` status code.

Otherwise, if the service returns a client `4XX` or service `5XX` status code, the service encountered an error and the resource shall not have been modified or created as a result of the operation.

### 7.6. PATCH (update)

To update a resource's properties, the service shall support the PATCH method.

The request body defines the changes to make to one or more properties in the resource that the request URI references. The PATCH request does not change any properties that are not in the request body. The service shall ignore OData annotations in the request body, such as `resource identifier`, `type`, and `ETag` properties. Services may accept a PATCH with an empty JSON object, which indicates that the service should make no changes to the resource.

When modification succeeds, the response may contain a representation of the updated resource. See Modification success responses.

To gain the protection semantics of an ETag, the service shall use the `If-Match` or `If-None-Match` header and not the `@odata.etag` property value for that protection.

The implementation may reject the update on certain properties based on its own policies and, in this case, not perform the requested update. For the following exception cases, services shall return the following HTTP status codes and other information:

<table>
<thead>
<tr>
<th>Exception case</th>
<th>The service returns</th>
</tr>
</thead>
</table>
| Modify several properties where one or more properties can never be updated. | • The HTTP `200 OK` status code.  
• A resource representation with a message annotation that lists the non-updatable properties.  
• The service may update other properties in the resource.  
For example, when a property is read-only, unknown, or unsupported. |

Redfish Specification

46

Published

Version 1.11.0
### Exception case

<table>
<thead>
<tr>
<th>Modify a single property that can never be updated. For example, a property that is read-only, unknown, or unsupported.</th>
<th>The service returns</th>
</tr>
</thead>
</table>
| • The HTTP [400 Bad Request](#) status code.  
• A resource representation with a message [annotation](#) that shows the non-updatable property. |

| Modify a resource or all properties that can never be updated. | The HTTP [405](#) status code. |

| A client PATCH request against a resource collection. | The HTTP [405](#) status code. |

| A client only provides OData annotations. | The HTTP [400 Bad Request](#) status code with the [NoOperation](#) message from the Base Message Registry or one of the [modification success responses](#). |

In the absence of outside changes to the resource, the PATCH operation should be idempotent, although the original ETag value may no longer match.

### 7.7. PATCH on array properties

The [Array properties](#) clause describes the three styles of array properties in a resource.

Within a PATCH request, the service shall accept `null` to remove an element, and accept an empty object `{}` to leave an element unchanged. Array properties that use the fixed or variable length style remove those elements, while array properties that use the rigid style replace removed elements with `null` elements. A service may indicate the maximum size of an array by padding `null` elements at the end of the array sequence.

When processing a PATCH request, the order of operations shall be:

- Modifications
- Deletions
- Additions

A PATCH request with fewer elements than in the current array shall remove the remaining elements of the array.

For example, a fixed length-style `Flavors` array indicates that the service supports a maximum of six
elements, by padding the array with null elements, with four populated.

```json
{
    "Flavors": [
        "Chocolate",
        "Vanilla",
        "Mango",
        "Strawberry",
        null,
        null
    ]
}
```

A client could issue the following PATCH request to remove Vanilla, replace Strawberry with Cherry, and add Coffee and Banana to the array, while leaving the other elements unchanged.

```json
{
    "Flavors": [
        {},
        null,
        {},
        "Cherry",
        "Coffee",
        "Banana"
    ]
}
```

After the PATCH operation, the resulting array is:

```json
{
    "Flavors": [
        "Chocolate",
        "Mango",
        "Cherry",
        "Coffee",
        "Banana",
        null
    ]
}
```
7.8. PUT (replace)

To completely replace a resource, services may support the PUT method. The service may add properties to the response resource that the client omits from the request body, the resource definition requires, or the service normally supplies.

The PUT operation should be idempotent in the absence of outside changes to the resource, with the possible exception that the operation might change ETag values.

When the replace operation succeeds, the response may contain a resource representation after the replacement occurs. See Modification success responses.

The following list contains the exception cases for PUT:

- If a service does not implement this method, the service shall return the HTTP 405 Method Not Allowed status code.
- Services may reject requests that do not include properties that the resource definition (schema) requires.
- If the client makes a PUT request against a resource collection, services should return the HTTP 405 Method Not Allowed status code.

7.9. POST (create)

To create a resource, services shall support the POST method on resource collections.

The POST request is submitted to the resource collection to which the new resource will belong. When the create operation succeeds, the response may contain the new resource representation. See Modification success responses.

The body of the create request contains a representation of the object to create. The service may ignore any service-controlled properties, such as Id, which would force the service to overwrite those properties. Additionally, the service shall set the Location header in the response to the URI of the new resource.

- Submitting a POST request to a resource collection is equivalent to submitting the same request to the Members property of that resource collection. Services that support the addition of Members to a resource collection shall support both forms.
  - For example, if a client adds a member to the resource collection at /redfish/v1/EventService/Subscriptions, it can send a POST request to either /redfish/v1/EventService/Subscriptions or /redfish/v1/EventService/Subscriptions/Members.
- If the service does not enable creation of resources, the service shall return the HTTP 405 Method Not Allowed status code.
- The POST operation shall not be idempotent.
• Services may allow the inclusion of \texttt{@Redfish.OperationApplyTime} property in the request body. See \textit{Operation apply time}.
• Services should return the HTTP 400 \texttt{Bad Request} status code for requests containing properties with the value \texttt{null}.

7.10. DELETE (delete)

To remove a resource, the service shall support the DELETE method. Resources \texttt{subordinate} to the resource removed by a DELETE method are typically removed, as the contents of subordinate resources are dependent on the parent resource. In some cases, related resources may also be relocated in the resource tree based on their definition and usage. Other resources in the resource tree may also be removed or incur side effects of a resource removal.

When the delete operation succeeds, the response may contain the resource representation after the deletion occurs. See \textit{Modification success responses}.

• If the resource can never be deleted, the service shall return the HTTP 405 \texttt{Method Not Allowed} status code.
• If the resource was already deleted, the service may return the HTTP 404 \texttt{Not Found} status code or a \textit{success code}.
• The service may allow the inclusion of the \texttt{@Redfish.OperationApplyTime} property in the request body. See \textit{Operation apply time}.

7.11. POST (Action)

Services shall support the POST method to send actions to resources.

• The POST operation may not be idempotent.
• Services may allow the inclusion of the \texttt{@Redfish.OperationApplyTime} property in the request body. See \textit{Operation apply time}.

To request actions on a resource, send the HTTP POST method to the URI of the action. The \texttt{target} property in the resource's \texttt{Actions property} shall contain the URI of the action. The URI of the action shall be in the format:

\texttt{ResourceUri/Actions/QualifiedActionName}

where

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{ResourceUri}</td>
<td>URI of the resource that supports the action.</td>
</tr>
<tr>
<td>Variable</td>
<td>Description</td>
</tr>
<tr>
<td>--------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Actions</td>
<td>Name of the property that contains the actions for a resource, as defined by</td>
</tr>
<tr>
<td>QualifiedActionName</td>
<td>Qualified name of the action. Includes the namespace.</td>
</tr>
</tbody>
</table>

To determine the available actions and the valid parameter values for those actions, clients can query a resource directly.

Clients provide parameters for the action as a JSON object within the request body of the POST operation. For information about the structure of the request and required parameters, see the Actions property clause. Some parameter information may require that the client examine the Redfish schema that corresponds to the resource.

The service may ignore unsupported parameters provided by the client. If an action does not have any required parameters, the service should accept an empty JSON object in the HTTP body for the action request.

To indicate the success or failure of the action request processing, the service may return a response with one of the following HTTP status codes and additional information:

<table>
<thead>
<tr>
<th>To indicate</th>
<th>HTTP status code</th>
<th>Additional information</th>
</tr>
</thead>
<tbody>
<tr>
<td>The action request succeeds, and the schema does not contain a response definition.</td>
<td>200 OK</td>
<td>The JSON message body, as described in Error responses, with a message that indicates success or any additional relevant messages. If the action was successfully processed and completed without errors, warnings, or other notifications for the client, the service should return the Success message from the Base Message Registry in the code property in the response body.</td>
</tr>
<tr>
<td>The action request succeeds, and the schema contains a response definition for the action.</td>
<td>200 OK</td>
<td>The JSON body in the response conforms to the action response defined in the schema.</td>
</tr>
<tr>
<td>The action request</td>
<td>201 Created</td>
<td>A Location response header set to the URI of the created resource. The JSON message body, as</td>
</tr>
<tr>
<td>To indicate</td>
<td>HTTP status code</td>
<td>Additional information</td>
</tr>
<tr>
<td>---------------------------------------------------------------------------</td>
<td>------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>succeeds, the schema does not contain a response definition, and a Location header in the response contains the URI of a created resource.</td>
<td></td>
<td>described in Error responses, with a message that indicates success or any additional relevant messages. If the action was successfully processed and completed without errors, warnings, or other notifications for the client, the service should return the Success message from the Base Message Registry in the code property in the response body.</td>
</tr>
<tr>
<td>The action request succeeds, the schema contains a response definition for the action, and a Location header in the response contains the URI of a created resource.</td>
<td>201 Created</td>
<td>A Location response header set to the URI of the created resource. The JSON body in the response conforms to the action response defined in the schema.</td>
</tr>
<tr>
<td>The action request may require extra time to process.</td>
<td>202 Accepted</td>
<td>A Location response header set to the URI of a task monitor.</td>
</tr>
<tr>
<td>The action request succeeds, and the schema does not contain a response definition.</td>
<td>204 No Content</td>
<td>No JSON message body.</td>
</tr>
<tr>
<td>The client did not provide all required</td>
<td>400 Bad Request</td>
<td>The response may contain a JSON object, as described in Error responses, which details the error or errors.</td>
</tr>
<tr>
<td>To indicate</td>
<td>HTTP status code</td>
<td>Additional information</td>
</tr>
<tr>
<td>----------------------------------------------------------------------------</td>
<td>------------------</td>
<td>----------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>The client provides a parameter that the service does not support, and the service does not ignore unsupported parameters.</td>
<td>400_Bad Request</td>
<td>The response may contain a JSON object, as described in Error responses, which details the error or errors.</td>
</tr>
<tr>
<td>An error was detected and the action request was not processed.</td>
<td>400 or greater</td>
<td>The response may contain a JSON object, as described in Error responses, which details the error or errors.</td>
</tr>
</tbody>
</table>

If an action requested by the client will have no effect, such as performing a reset of a ComputerSystem where the ResetType parameter is set to On and the ComputerSystem is already On, the service should respond with the HTTP 200_OK status code and return the NoOperation message from the Base Message Registry.

Example successful action response:

```json
{
   "error": {
      "code": "Base.1.0.Success",
      "message": "Successfully Completed Request",
      "@Message.ExtendedInfo": [
         {
            "@odata.type": "#Message.v1_0_0.Message",
            "MessageId": "Base.1.0.Success",
            "Message": "Successfully Completed Request",
            "Severity": "OK",
            "Resolution": "None"
         }
      ]
   }
}
```
7.12. Operation apply time

Services may accept the `@Redfish.OperationApplyTime` annotation in the POST (create), DELETE (delete), or POST (action) request body. This annotation enables the client to control when an operation is carried out.

For example, if the client wants to delete a particular Volume resource, but can only safely do so when a reset occurs, the client can use this annotation to instruct the service to delete the Volume on the next reset.

If multiple operations are pending, the service shall process them in the order in which the service receives them.

Services that support the `@Redfish.OperationApplyTime` annotation for create and delete operations on a resource collection shall include the `@Redfish.OperationApplyTimeSupport` response annotation for the resource collection.

The following example response for a resource collection supports the `@Redfish.OperationApplyTime` annotation in the create and delete requests:

```json
{
    "@odata.id": "/redfish/v1/Systems/1/Storage/SATAEmbedded/Volumes",
    "@odata.type": "#VolumeCollection.VolumeCollection",
    "Name": "Storage Volume Collection",
    "Description": "Storage Volume Collection",
    "Members@odata.count": 2,
    "Members": [
    {
        "@odata.id": "/redfish/v1/Systems/1/Storage/SATAEmbedded/Volumes/1"
    },
    {   
        "@odata.id": "/redfish/v1/Systems/1/Storage/SATAEmbedded/Volumes/2"
    }
    ],
    "@Redfish.OperationApplyTimeSupport": {
        "@odata.type": "#Settings.v1_2_0.OperationApplyTimeSupport",
        "SupportedValues": [ "Immediate", "OnReset" ]
    }
}
```

In the previous example, a client can annotate their create request body on the VolumeCollection itself, or a delete operation on the Volumes within the VolumeCollection.

The following sample request deletes a Volume on the next reset:
DELETE /redfish/v1/Systems/1/Storage/SATAEmbedded/Volumes/2 HTTP/1.1
Content-Type: application/json;charset=utf-8
Content-Length: <computed length>
OData-Version: 4.0

{
    "@Redfish.OperationApplyTime": "OnReset"
}

Services that support the @Redfish.OperationApplyTime annotation for an action shall include the @Redfish.OperationApplyTimeSupport response annotation for the action.

The following example response for a ComputerSystem resource supports the @Redfish.OperationApplyTime annotation in the reset action request:

```
{
    "@odata.id": "/redfish/v1/Systems/1",
    "@odata.type": "#ComputerSystem.v1_5_0.ComputerSystem",
    "Actions": {
        "#ComputerSystem.Reset": {
            "@odata.id": "/redfish/v1/Systems/1/Actions/ComputerSystem.Reset",
            "ResetType@Redfish.AllowableValues": [
                "On",
                "ForceOff",
                "ForceRestart",
                "Nmi",
                "ForceOn",
                "PushPowerButton"
            ],
            "@Redfish.OperationApplyTimeSupport": {
                "@odata.type": "#Settings.v1_2_0.OperationApplyTimeSupport",
                "SupportedValues": [ "Immediate", "AtMaintenanceWindowStart" ],
                "MaintenanceWindowStartTime": "2017-05-03T23:12:37-05:00",
                "MaintenanceWindowDurationInSeconds": 600,
                "MaintenanceWindowResource": {
                    "@odata.id": "/redfish/v1/Systems/1"
                }
            }
        }
    }
}
...
```

In the previous example, a client can annotate their reset action request body on the ComputerSystem
in the payload.

The following sample request completes a reset at the start of the next maintenance window:

```plaintext
POST /redfish/v1/Systems/1/Actions/ComputerSystem.Reset HTTP/1.1
Content-Type: application/json;charset=utf-8
Content-Length: <computed length>
OData-Version: 4.0

{
    "ResetType": "ForceRestart",
    "@Redfish.OperationApplyTime": "AtMaintenanceWindowStart"
}
```

Services that support the @Redfish.OperationApplyTime annotation for a resource collection or action shall create a task, and respond with the HTTP 202 Accepted status code with a Location header set to the URI of a task monitor, if the client's request body contains @Redfish.OperationApplyTime in the request.

The Settings Redfish schema defines the structure of the @Redfish.OperationApplyTimeSupport object and the @Redfish.OperationApplyTime annotation value.

### 7.13. Deep operations

Implementations may support operations that modify the current resource as well as subordinate resources. These operations are known as deep operations. They give the client the ability to modify more than one resource with a single operation.

The following types of deep operations are defined by this specification:

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Deep PATCH</td>
<td>Modify a resource and one or more subordinate resources.</td>
<td>Modify a ComputerSystem resource as well as subordinate Storage and NetworkAdapter resources.</td>
</tr>
<tr>
<td>(update)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Deep POST</td>
<td>Create multiple resources in a resource collection.</td>
<td>Create ManagerAccount resources.</td>
</tr>
<tr>
<td>(create)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Services that support deep PATCH for updating resources shall set the value of the DeepPATCH property in the DeepOperations property in the ProtocolFeaturesSupported property.
within the service root to true.

- Services that support deep POST for creating resources shall set the value of the DeepPOST property in the DeepOperations property in the ProtocolFeaturesSupported property within the service root to true.
- The Members property in resource collections shall not be removed when using a deep PATCH.
- Action URIs shall not support deep POST operations.
- If the service supports deep operations, the MaxLevels property in the DeepOperations property in the ProtocolFeaturesSupported property in the service root shall indicate the maximum number of levels that the service supports for deep operations.
- To request deep operations on a resource, send the HTTP method to the deep operation URI of the resource. The URI for deep operations on any resource shall be in the format: ResourceUri.Deep.
- The schema used for validating the root level of the request body shall be the schema of the resource in the resource URI.
  - The subordinate resources included in the request body shall be validated against their corresponding schema.

The body of deep operations contains the resource being modified as well as the subordinate resources being modified. This resource can be a collection or a single instance. These resources could be subordinate resources, subordinate resource collections, or subordinate members of resource collections. The client can omit properties from the request such as those it does not want to modify or that the service controls. Requests that include references to multiple instances, such as members of a collection, shall include the Members property as part of the request body.

In order to determine which members of subordinate resource collections are to be modified by a deep PATCH, services shall use the @odata.id property provided by the client to identify the member of the resource collection to be modified.

Clients may provide the @odata.etag property in subordinate resources being modified by a deep PATCH. If the If-Match or If-None-Match header is specified in the request, the service shall compare the ETag in the request header with the ETag of the resource specified by the URI. If this check passes, then the operation can proceed using the @odata.etag values contained in the body of the subordinate resources. The operation on each subordinate resource shall be performed independently in this case, where some subordinate values that pass the condition check proceed and the resources that fail do not proceed. In this case, annotated extended information shall be included in the subordinate resource representation of the response.

Failure semantics for deep operations are similar to that of other operations of similar type. If any properties in a deep PATCH operation succeeded, then the result is a 200 OK with the results returned in the response, and the service should include extended information indicating warnings or errors. For a deep POST operation, if any member of the collection was created then a 201 Created shall be returned, and any members that were not created should have extended information in their place holders with sufficient identifying information, such as returning all of the properties provided in the POST request.
body for that member, as well as extended information indicating why the creation was not successful. When performing a deep POST, the value of the Location header shall be that of one of the URIs created and should be that of one of the least subordinate URIs, such as that of a ComputerSystem resource and not one of the devices subordinate to the ComputerSystem resource.

Deep POST shall not be allowed on the Sessions Collection.

The following is an example of a deep PATCH showing the RoleCollection resource with two members being modified:

```
PATCH /redfish/v1/AccountService/Roles.Deep HTTP/1.1
Content-Type: application/json;charset=utf-8
Content-Length: <computed length>
OData-Version: 4.0

{
  "Members": [
    {
      "@odata.id": "/redfish/v1/AccountService/Roles/OperatorRestricted",
      "AssignedPrivileges": [
        "Login",
        "ConfigureComponents"
      ]
    },
    {
      "@odata.id": "/redfish/v1/AccountService/Roles/ReadOnlyRestricted",
      "AssignedPrivileges": [
        "Login"
      ]
    }
  ]
}
```

The following is an example of a deep POST showing the RoleCollection resource with two members being created:

```
POST /redfish/v1/AccountService/Roles.Deep HTTP/1.1
Content-Type: application/json;charset=utf-8
Content-Length: <computed length>
OData-Version: 4.0

{
  "Members": [
    {
      "@odata.id": "/redfish/v1/AccountService/Roles/OperatorRestricted",
      "AssignedPrivileges": [
        "Login",
        "ConfigureComponents"
      ]
    },
    {
      "@odata.id": "/redfish/v1/AccountService/Roles/ReadOnlyRestricted",
      "AssignedPrivileges": [
        "Login"
      ]
    }
  ]
}
```
"RoleId": "OperatorRestricted",
"AssignedPrivileges": [
 "Login",
 "ConfigureComponents"
],
},
{
 "RoleId": "ReadOnlyRestricted",
 "AssignedPrivileges": [
  "Login"
]
}
]

The following is an example of a deep PATCH showing a ComputerSystem resource where there is a request to modify its assert tag and BIOS settings:

```json
PATCH /redfish/v1/Systems/47832.Deep HTTP/1.1
Content-Type: application/json;charset=utf-8
Content-Length: <computed length>
OData-Version: 4.0

{
 "AssetTag": "Inventory Tag 12394783431",
 "Bios": {
  "@odata.id": "/redfish/v1/Systems/47832/Bios",
  "@Redfish.Settings": {
   "SettingsObject": {
    "@odata.id": "/redfish/v1/Systems/1/Bios/SD",
    "Attributes": {
     "AdminPhone": "(123) 456-789",
     "BootMode": "Uefi"
    }
   }
  }
 }
}
```

The following shows a deep PATCH operation with ETAGs:

```json
PATCH /redfish/v1/AccountService/Roles.Deep HTTP/1.1
Content-Type: application/json;charset=utf-8
```

DSP0266

Redfish Specification

Version 1.11.0

59
The following is an example of a partial failure of a deep PATCH operation with ETags:

HTTP/1.1 200 OK
Content-Type: application/json;charset=utf-8
Content-Length: <computed length>
ETag: <Resource collection ETag>
OData-Version: 4.0

{
   "Members": [
      {
         "@odata.id": "/redfish/v1/AccountService/Roles/OperatorRestricted",
         "@odata.ETag": "ABCDEFG",
         "AssignedPrivileges": ["Login",
                              "ConfigureComponents"
                        ]
      },
      {
         "@odata.id": "/redfish/v1/AccountService/Roles/ReadOnlyRestricted",
         "@odata.ETag": "ABCDEFG",
         "AssignedPrivileges": ["Login"
                  ]
      }
   ]
}
8. Service responses

This clause describes the responses that Redfish Services can send to clients.

8.1. Response headers

HTTP defines headers for use in response messages. The following table defines those headers and their requirements for Redfish Services:

- Redfish Services shall return the HTTP 1.1 Specification-defined headers if the value in the Required column is Yes.
- Redfish Services should return the HTTP 1.1 Specification-defined headers if the value in the Required column is No.
- Redfish clients shall be able to both understand and process all the HTTP 1.1 Specification-defined headers.

<table>
<thead>
<tr>
<th>Header</th>
<th>Required</th>
<th>Supported values</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Access-Control-Allow-Origin</td>
<td>Yes</td>
<td>W3C CORS, Section 5.1</td>
<td>Prevents or allows requests based on originating domain. Prevents CSRF attacks.</td>
</tr>
<tr>
<td>Allow</td>
<td>Yes</td>
<td>POST, PUT, PATCH, DELETE, GET, HEAD</td>
<td>Shall be returned with the HTTP 405 (Method Not Allowed) status code to indicate the valid methods for the request URI. Shall be returned with any GET or HEAD operation to indicate the other allowable operations for this resource.</td>
</tr>
<tr>
<td>Header</td>
<td>Required</td>
<td>Supported values</td>
<td>Description</td>
</tr>
<tr>
<td>-----------------</td>
<td>----------</td>
<td>------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Cache-Control</td>
<td>Yes</td>
<td>RFC7234</td>
<td>Shall be supported and indicates whether a response can or cannot be cached.</td>
</tr>
<tr>
<td>Content-Encoding</td>
<td>No</td>
<td>RFC7231</td>
<td>The encoding that has been performed on the media type.</td>
</tr>
<tr>
<td>Content-Length</td>
<td>No</td>
<td>RFC7231</td>
<td>The size of the message body. An optional means of indicating size of the body uses Transfer-Encoding: chunked, that does not use the Content-Length header. If a service does not support Transfer-Encoding and needs Content-Length instead, the service shall respond with the HTTP 411 Length Required status code.</td>
</tr>
<tr>
<td>Content-Type</td>
<td>Yes</td>
<td>RFC7231</td>
<td>The message body's representation type.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Services shall specify a Content-Type of application/json when returning resources as JSON.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Services shall specify a Content-Type of application/xml when returning metadata as XML.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Services shall specify a Content-Type of application/yaml or application/vnd.oai.openapi when returning OpenAPI schema as YAML.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Services shall specify a Content-Type of text/event-stream when returning an SSE stream.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>; charset=utf-8 shall be appended to the Content-Type if specified in the</td>
</tr>
<tr>
<td>Header</td>
<td>Required</td>
<td>Supported values</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>----------</td>
<td>------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>chosen media-type in the <code>Accept</code> header for the request.</td>
</tr>
<tr>
<td>ETag</td>
<td>Conditional</td>
<td>RFC7232</td>
<td>An identifier for a specific version of a resource, often a message digest. The <code>ETag</code> header shall be included on responses to GETs of ManagerAccount resources.</td>
</tr>
<tr>
<td>Link</td>
<td>Yes</td>
<td>See Link header</td>
<td>Link headers shall be returned, as described in the Link headers clause.</td>
</tr>
<tr>
<td>Location</td>
<td>Conditional</td>
<td>RFC7231</td>
<td>A URI that requests a representation of the resource. Shall be returned upon creation of a resource. Location and X-Auth-Token shall be included on responses that create user sessions.</td>
</tr>
<tr>
<td>Max-Forwards</td>
<td>No</td>
<td>RFC7231</td>
<td>Limits gateway and proxy hops. Prevents messages from remaining in the network indefinitely.</td>
</tr>
<tr>
<td>OData-Version</td>
<td>Yes</td>
<td>4.0</td>
<td>The OData version of the payload to which the response conforms.</td>
</tr>
<tr>
<td>Retry-After</td>
<td>No</td>
<td>RFC7231, Section 7.1.3</td>
<td>Informs a client how long to wait before requesting the task information again.</td>
</tr>
<tr>
<td>Server</td>
<td>No</td>
<td>RFC7231</td>
<td>A product token and its version. Multiple product tokens may be listed.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td><strong>Note:</strong> Previous versions of the Specification marked this header as required. This has been changed as no use cases for requiring it have been identified.</td>
</tr>
<tr>
<td>Via</td>
<td>No</td>
<td>RFC7230</td>
<td>Defines the network hierarchy and recognizes message loops. Each pass inserts its own Via header.</td>
</tr>
<tr>
<td>Header</td>
<td>Required</td>
<td>Supported values</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------</td>
<td>----------</td>
<td>------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>WWW-Authenticate</td>
<td>Yes</td>
<td>RFC7617</td>
<td>Required for Basic and other optional authentication mechanisms. For details, see the Security details clause.</td>
</tr>
<tr>
<td>X-Auth-Token</td>
<td>Yes</td>
<td>Opaque encoded octet strings</td>
<td>Contains the authentication token for user sessions. The token value shall be indistinguishable from random.</td>
</tr>
</tbody>
</table>

### 8.2. Link header

The [Link header](http://www.example.com) provides metadata information on the accessed resource in response to a HEAD or GET request. The metadata information can include hyperlinks from the resource and JSON Schemas that describe the resource.

The following example shows the Link headers for a ManagerAccount with an Administrator role, in addition to a Settings annotation:

```plaintext
Link: </redfish/v1/AccountService/Roles/Administrator>; path=/Links/Role
Link: <http://redfish.dmtf.org/schemas/Settings.json>
Link: </redfish/v1/JsonSchemas/ManagerAccount.v1_0_2.json>; rel=describedby
```

- The first Link header is an example of a hyperlink that comes from the resource. It describes hyperlinks within the resource. This type of header is outside the scope of this specification.
- The second Link header is an example of an Annotation Link header as it references the JSON Schema that describes the annotation and does not have rel=describedby. This example references the public copy of the annotation on the DMTF's Redfish schema repository.
- The third Link header is an example for the JSON Schema that describes the actual resource.
  - Note that the URL can reference an unversioned JSON Schema because the @odata.type in the resource indicates the appropriate version, or reference the versioned JSON Schema, which according to previous normative statements need to match the version in the @odata.type property of the resource.

A Link header containing rel=describedby shall be returned on GET and HEAD requests. If the referenced JSON Schema is a versioned schema, it shall match the version contained in the value of the @odata.type property returned in this resource.

A Link header satisfying annotations should be returned on GET and HEAD requests.
8.3. Status codes

HTTP defines status codes that appear in responses. The status codes themselves provide general information about how the request was processed, such as whether the request was successful, if the client provided bad information, or the service encountered an error when performing the request.

- When the service returns a status code in the 4XX or 5XX range, services should return an extended error response in the response body to provide the client more meaningful and deterministic error semantics.
- When the service returns a status code in the 2XX range and the response contains a representation of a resource, services may use extended information to convey additional information about the resource.
- Extended error messages shall not provide privileged information when authentication failures occur.

*Note:* For security implications of extended errors, See [Security details](#).

The following table lists HTTP status codes that have meaning or usage defined for a Redfish Service, or are otherwise referenced by this specification. Other codes may be returned by the service as appropriate, and their usage is implementation-specific. For usage and additional requirements imposed by this specification, see the Description column.

- Clients shall understand and be able to process the status codes in the following table as defined by the HTTP 1.1 Specification and constrained by additional requirements defined by this specification.
- Services shall respond with the status codes in the following table as defined in description column.
- Redfish Services should not return the HTTP 100 status code. Using the HTTP protocol for a multipass data transfer should be avoided, except for the upload of extremely large data.
- If no other status code in the 4XX range is appropriate for client-side errors, the default status code should be the HTTP 400 Bad Request status code.
- If no other status code in the 5XX range is appropriate for service-side errors, the default status code should be the HTTP 500 Internal Server Error status code.

<table>
<thead>
<tr>
<th>HTTP status code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>200 OK</td>
<td>Request completed successfully and includes a representation in its body.</td>
</tr>
<tr>
<td>201 Created</td>
<td>Request to create a resource completed successfully. The Location header shall be set to the canonical URI for the newly created resource. The response body may include a representation of the newly created resource.</td>
</tr>
<tr>
<td>202 Accepted</td>
<td>Request has been accepted for processing but the processing has not been</td>
</tr>
<tr>
<td>HTTP status code</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>200 OK</td>
<td>The request succeeded. The Location header shall be set to the URI of a task monitor that can later be queried to determine the status of the operation. The response body may include a representation of the Task resource.</td>
</tr>
<tr>
<td>204 No Content</td>
<td>The request succeeded, but no content is being returned in the body of the response.</td>
</tr>
<tr>
<td>301 Moved</td>
<td>Requested resource resides under a different URI.</td>
</tr>
<tr>
<td>302 Found</td>
<td>Requested resource resides temporarily under a different URI.</td>
</tr>
<tr>
<td>304 Not Modified</td>
<td>Service has performed a conditional GET request where access is allowed but the resource content has not changed. Either or both the If-Modified-Since and If-None-Match headers initiate conditional requests to save network bandwidth if no change has occurred. See HTTP 1.1, sections 14.25 and 14.26.</td>
</tr>
<tr>
<td>400 Bad Request</td>
<td>Request could not be processed because it contains invalid information, such as an invalid input field, or is missing a required value. The response body shall return an extended error as defined in the Error responses clause.</td>
</tr>
<tr>
<td>401 Unauthorized</td>
<td>Authentication credentials included with this request are missing or invalid.</td>
</tr>
<tr>
<td>403 Forbidden</td>
<td>Service recognized the credentials in the request but those credentials do not possess authorization to complete this request. This code is also returned when the user credentials provided need to be changed before access to the service can be granted. For details, see the Security details clause.</td>
</tr>
<tr>
<td>404 Not Found</td>
<td>Request specified a URI of a resource that does not exist.</td>
</tr>
<tr>
<td>405 Method Not Allowed</td>
<td>HTTP verb in the request, such as DELETE, GET, HEAD, POST, PUT, or PATCH, is not supported for this request URI. The response shall include an Allow header that provides a list of methods that the resource identified by the URI in the client request supports.</td>
</tr>
<tr>
<td>406 Not Acceptable</td>
<td>Accept header was specified in the request and the resource identified by this request cannot generate a representation that corresponds to one of the media types in the Accept header.</td>
</tr>
<tr>
<td>409 Conflict</td>
<td>Creation or update request could not be completed because it would cause a conflict in the current state of the resources that the platform supports.</td>
</tr>
<tr>
<td>HTTP status code</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>-------------</td>
</tr>
<tr>
<td>410 Gone</td>
<td>Requested resource is no longer available at the service and no forwarding address is known. This condition is expected to be considered permanent. Clients with hyperlink editing capabilities should delete references to the URI in the client request after user approval. If the service does not know or cannot determine whether the condition is permanent, client should use the HTTP 404 Not Found status code. This response is cacheable unless otherwise indicated.</td>
</tr>
<tr>
<td>411 Length Required</td>
<td>Request did not use the Content-Length header to specify the length of its content but perhaps used the Transfer-Encoding: chunked header instead. The addressed resource requires the Content-Length header.</td>
</tr>
<tr>
<td>412 Precondition Failed</td>
<td>Precondition check, such as check of the OData-Version, If-Match, or If-Not-Modified header, failed.</td>
</tr>
<tr>
<td>415 Unsupported Media Type</td>
<td>Request specifies a Content-Type for the body that is not supported.</td>
</tr>
<tr>
<td>428 Precondition Required</td>
<td>Request did not provide the required precondition, such as an If-Match or If-None-Match header.</td>
</tr>
<tr>
<td>431 Request Header Field Too Large</td>
<td>Service is unwilling to process the request because either an individual header field or the collection of all header fields are too large.</td>
</tr>
<tr>
<td>500 Internal Server Error</td>
<td>Service encountered an unexpected condition that prevented it from fulfilling the request. The response body shall return an extended error as defined in the Error responses clause.</td>
</tr>
<tr>
<td>501 Not Implemented</td>
<td>Service does not currently support the functionality required to fulfill the request. This response is appropriate when the service does not recognize the request method and cannot support the method for any resource.</td>
</tr>
<tr>
<td>503 Service Unavailable</td>
<td>Service currently cannot handle the request due to temporary overloading or maintenance of the service. A service may use this response to indicate that the request URI is valid but the service is performing initialization or other maintenance on the resource. It may also use this response to indicate the</td>
</tr>
</tbody>
</table>
HTTP status code | Description
--- | ---
| | service itself is undergoing maintenance, such as finishing initialization steps after reboot of the service.
507 Insufficient Storage | Service cannot build the response for the client due to the size of the response.

8.4. OData metadata responses

OData metadata describes resources, resource collections, capabilities, and service-dependent behavior to generic OData consumers with no specific understanding of this specification. Clients are not required to request metadata if they already have sufficient understanding of the target service. For example, clients are not required to request metadata to request and interpret a JSON representation of a resource that this specification defines.

A client can access the OData metadata at the /redfish/v1/$metadata URI.

A client can access the OData service document at the /redfish/v1/odata URI.

8.4.1. OData $metadata

The OData metadata describes top-level service resources and resource types according to OData Common Schema Definition Language. The OData metadata is represented as an XML document with an Edmx root element in the http://docs.oasis-open.org/odata/ns/edmx namespace with an OData version attribute set to 4.0.

The service shall use the application/xml or application/xml;charset=utf-8 MIME types to return the OData metadata document as an XML document.

```
<edmx:Edmx xmlns:edmx="http://docs.oasis-open.org/odata/ns/edmx" Version="4.0">
  <!-- edmx:Reference and edmx:Schema elements go here -->
</edmx:Edmx>
```

8.4.1.1. Referencing other schemas

The OData metadata shall include the namespaces for each of the Redfish resource types, along with the RedfishExtensions.v1_0_0 namespace.

These references may use either:
The standard URI for the published Redfish schema definitions, such as on http://redfish.dmtf.org/schemas.

- A URI to a local version of the Redfish schema.

```xml
<edmx:Reference Uri="http://redfish.dmtf.org/schemas/v1/ServiceRoot_v1.xml">
    <edmx:Include Namespace="ServiceRoot"/>
    <edmx:Include Namespace="ServiceRoot.v1_0_0"/>
</edmx:Reference>
...

<edmx:Reference Uri="http://redfish.dmtf.org/schemas/v1/VirtualMedia_v1.xml">
    <edmx:Include Namespace="VirtualMedia"/>
    <edmx:Include Namespace="VirtualMedia.v1_0_0"/>
</edmx:Reference>
<edmx:Reference Uri="http://redfish.dmtf.org/schemas/v1/RedfishExtensions_v1.xml">
    <edmx:Include Namespace="RedfishExtensions.v1_0_0" Alias="Redfish"/>
</edmx:Reference>
```

The service’s OData metadata document shall include an EntityContainer that defines the top-level resources and resource collections.

8.4.1.2. Referencing OEM extensions

The OData metadata document may reference additional schema documents that describe OEM-specific extensions that the service uses.

For example, the OData metadata document may reference custom types for additional resource collections.

```xml
<edmx:Reference Uri="http://contoso.org/Schema/CustomTypes">
    <edmx:Include Namespace="CustomTypes"/>
</edmx:Reference>
```

8.4.2. OData service document

The OData service document serves as a top-level entry point for generic OData clients. More information about the OData service document can be found in the OData JSON Format Specification.

```json
{
    "@odata.context": "/redfish/v1/$metadata",
    "value": [}
The service shall use the `application/json` MIME type to return the OData service document as a JSON object.

The JSON object shall contain the `@odata.context` context property set to `/redfish/v1/$metadata`.

The JSON object shall include a `value` property set to a JSON array that contains an entry for the Service Root and each resource that is a direct child of the Service Root.

Each JSON object entry includes:

<table>
<thead>
<tr>
<th>Property</th>
<th>Defines</th>
</tr>
</thead>
<tbody>
<tr>
<td>name</td>
<td>User-friendly resource name of the resource.</td>
</tr>
<tr>
<td>kind</td>
<td>Type of resource. Value is Singleton for all cases defined by Redfish.</td>
</tr>
<tr>
<td>url</td>
<td>Relative URL for the top-level resource.</td>
</tr>
</tbody>
</table>

### 8.5. Resource responses

Services use the `application/json` MIME type to return resources and resource collections as JSON payloads. A service shall not break responses for a single resource into multiple results.

The format of these payloads is defined by the Redfish schema. For rules about the Redfish schema and how it maps to JSON payloads, see the [Data model](#) and [Schema definition languages](#) clauses.
8.6. Error responses

HTTP status codes often do not provide enough information to enable deterministic error semantics. For example, if a client makes a PATCH call and some properties do not match while others are not supported, the HTTP 400 Bad Request status code does not tell the client which values are in error. Error responses provide the client more meaningful and deterministic error semantics.

To provide the client with as much information about the error as possible, a Redfish Service may provide multiple error responses in the HTTP response. Additionally, the service may provide Redfish standardized errors, OEM-defined errors, or both, depending on the implementation's ability to convey the most useful information about the underlying error.

An extended error response, which is a single JSON object, defines the error responses, with an error property, which contains the following properties.

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>code</td>
<td>String. Defines a MessageId from the Message Registry.</td>
</tr>
<tr>
<td>message</td>
<td>Displays a human-readable error message that corresponds to the message in the Message Registry.</td>
</tr>
<tr>
<td>@Message.ExtendedInfo</td>
<td>Displays an array of message objects. Describes one or more error messages.</td>
</tr>
</tbody>
</table>

See the Schema definition languages clause for references to the schema definitions of the error response payload.

The @Message.ExtendedInfo property should be present in all error responses. If the @Message.ExtendedInfo property is present, all information necessary to process the error should be provided in the @Message.ExtendedInfo property. Clients should look for the @Message.ExtendedInfo property for error processing first, and fallback on the code and message properties if @Message.ExtendedInfo is not present.

The following sample error response contains two messages in the @Message.ExtendedInfo property that describe two different errors. The message described by the code and message properties do not provide actionable information for the client.

```json
{
    "error": {
        "code": "Base.1.7.GeneralError",
        "message": "A general error has occurred. See Resolution for information on how to resolve the error."
    }
}
```
9. Data model

One of the key tenets of Redfish is the separation of protocol from the data model. This separation makes the data both transport and protocol agnostic. By concentrating on the data transported in the payload of the protocol (in HTTP, it is the HTTP body), Redfish can also define the payload in any encoding and the data model is intended to be schema-language agnostic. While Redfish uses the JSON data-interchange format, Redfish provides a common encoding type that ensures property naming conventions that make development easier in JavaScript, Python, and other languages. This encoding type helps the Redfish data model be more easily accessible in modern tools and programming environments.
This clause describes common data model, resource, and Redfish schema requirements.

9.1. Resources

A resource is a single entity. Services use the application/json MIME type to return resources as JSON payloads.

Each resource shall be strongly typed and defined in a Redfish schema document, and identified in the response payload by a unique type identifier property.

Responses for a single resource shall contain the following properties:

- @odata.id
- @odata.type
- Id
- Name

Responses may also contain other properties defined within the schema for that resource type. Responses shall not include any properties not defined by that resource type.

9.2. Resource collections

A resource collection is a set of resources that share the same schema definition. Services use the application/json MIME type to return resource collections as JSON payloads.

Resource collection responses shall contain the following properties:

- @odata.id
- @odata.type
- Name
- Members
- Members@odata.count

Responses for resource collections may contain the following properties:

- @odata.context
- @odata.etag
- Description
- Members@odata.nextLink
- Oem

Responses for resource collections shall not contain any other properties with the exception of payload.
9.3. OEM resources

OEMs and other third parties can extend the Redfish data model by creating resource types. This is accomplished by defining an OEM schema for each resource type, and connecting instances of those resources to the resource tree.

Companies, OEMs, and other organizations use the Oem property in resources, the links property, and actions to define additional properties, hyperlinks, and actions for standard Redfish resources.

While the information and semantics of these extensions are outside of the standard, the schema representing the data, the resource itself, and the semantics around the protocol shall conform to the requirements in this specification. OEMs are encouraged to follow the design tenets and naming conventions in this specification when defining OEM resources or properties.

9.4. Common data types

The following clause details the data types found throughout the Redfish data model.

9.4.1. Primitive types

The following are the primitive data types in the data model:

<table>
<thead>
<tr>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Boolean</td>
<td>A variable with a value of true or false.</td>
</tr>
<tr>
<td>Number</td>
<td>A number with optional decimal point or exponent. Number properties may restrict the representation to an integer or a number with decimal point.</td>
</tr>
<tr>
<td>String</td>
<td>A sequence of characters enclosed with double quotes (&quot;).</td>
</tr>
<tr>
<td>Array</td>
<td>A comma-separated set of the previous types enclosed with square braces ([ and ]). See the Array properties clause.</td>
</tr>
<tr>
<td>Object</td>
<td>A set of properties enclosed with curly braces ({ and }). See the Structured properties clause.</td>
</tr>
<tr>
<td>Null</td>
<td>The null value, which the service uses when it is unable to determine the property value, or if the schema has requirements for using null for other conditions.</td>
</tr>
</tbody>
</table>

When receiving values from the client, services should support other valid representations of the data in the specified JSON type. In particular, services should support valid integer and decimal values in...
exponential notation and integer values that contain a decimal point with no non-zero trailing digits.

### 9.4.2. GUID and UUID values

Globally Unique Identifier (GUID) and Universally Unique Identifier (UUID) values are unique identifier strings and shall use the format:

```
([0-9a-f]{8}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{12})
```

### 9.4.3. Date-Time values

Date-Time values are strings according to the ISO 8601 extended format, including the time offset or UTC suffix.

Date-Time values shall use the format:

```
YYYY-MM-DDTh:mm:ss[.SSS](Z|((+|-)HH:MM))
```

<table>
<thead>
<tr>
<th>Variable or separator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>YYYY</td>
<td>Four-digit year.</td>
</tr>
<tr>
<td>MM</td>
<td>Two-digit month (1 to 12).</td>
</tr>
<tr>
<td>DD</td>
<td>Two-digit day (1 to 31).</td>
</tr>
<tr>
<td>T</td>
<td>Time separator. Shall be a capital T.</td>
</tr>
<tr>
<td>hh</td>
<td>Two-digit hour (0 to 23).</td>
</tr>
<tr>
<td>mm</td>
<td>Two-digit minute (0 to 59).</td>
</tr>
<tr>
<td>ss</td>
<td>Two-digit second (0 to 59).</td>
</tr>
<tr>
<td>SSS</td>
<td>Optional. Decimal fraction of a second. Shall be one or more digits where the number of digits implies the precision.</td>
</tr>
<tr>
<td>Z</td>
<td>Zero offset indicator. Shall be a capital Z.</td>
</tr>
<tr>
<td>HH</td>
<td>Two-digit hour offset (0 to 23).</td>
</tr>
<tr>
<td>MM</td>
<td>Two-digit minute offset (0 to 59).</td>
</tr>
</tbody>
</table>

For example, 2015-03-13T04:14:33+06:00 represents March 13, 2015 at 4:14:33 with a +06:00 time offset.
When the time of day is unknown or serves no purpose, the service shall report 00:00:00Z for the time of day value.

9.4.4. Duration values

Duration values are strings according to the ISO 8601 duration format, with the exception of not expressing a representation for years, months, or weeks. Duration values shall use the format:

\[ P[dD][T[hH][mM][s[.f]S]] \]

where

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>d</td>
<td>Number of days.</td>
</tr>
<tr>
<td>h</td>
<td>Number of hours.</td>
</tr>
<tr>
<td>m</td>
<td>Number of minutes.</td>
</tr>
<tr>
<td>s</td>
<td>Number of seconds.</td>
</tr>
<tr>
<td>f</td>
<td>Fractional seconds.</td>
</tr>
</tbody>
</table>

Each field is optional and may contain more than one digit.

For example, the following values represent the following durations:

<table>
<thead>
<tr>
<th>Value</th>
<th>Duration</th>
</tr>
</thead>
<tbody>
<tr>
<td>P90D</td>
<td>Ninety days.</td>
</tr>
<tr>
<td>P3D</td>
<td>Three days.</td>
</tr>
<tr>
<td>PT6H</td>
<td>Six hours.</td>
</tr>
<tr>
<td>PT10S</td>
<td>Ten seconds.</td>
</tr>
<tr>
<td>PT0.001S</td>
<td>0.001 seconds.</td>
</tr>
<tr>
<td>PT1H30M</td>
<td>One hour and 30 minutes.</td>
</tr>
</tbody>
</table>

**DEPRECATED:** Duration values shall use the format: \[ P[yY][mM][wW][dD][T[hH][mM][s[.f]S]] \]. This definition allows for specifying years,
months, and weeks. ISO 8601 does not specify an exact value for the duration of a year or of a month, which introduces interoperability challenges.

9.4.5. Reference properties

Reference properties provide a reference to another resource in the data model. Reference properties are JSON objects that contain an @odata.id property. The @odata.id property value is the URI of the referenced resource.

9.4.6. Non-resource reference properties

Non-resource reference properties provide a reference to services or documents that are not Redfish-defined resources. These properties shall include the Uri term in their property name. For example, AssemblyBinaryDataUri in the Assembly schema. The access protocol and data format of the referenced URI may be defined in schema for that property. Non-resource reference properties that refer to local HTTP/S targets shall follow the Redfish protocol, including use of Redfish sessions and access control, unless otherwise specified by the property definition in schema.

9.4.7. Array properties

Array properties contain a set of values or objects, and appear as JSON arrays within a response body. Array elements shall all contain values of the same data type.

There are three styles of arrays, regardless of the data type of the elements:

<table>
<thead>
<tr>
<th>Array style</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fixed length</td>
<td>Contains a static number of elements. The property definition sets or the implementation chooses the size of the array.</td>
</tr>
<tr>
<td>Variable length</td>
<td>Contains a variable number of elements. The array size is not specified and the size varies among instances. The array size may change. This array style is the most common style.</td>
</tr>
<tr>
<td>Rigid</td>
<td>The array index is meaningful. When elements are added to or removed from the array, the elements do not change their position, or index, in the array. An element that is removed from a rigid array shall be replaced by a null element and all other elements shall remain at their current index. Empty elements in a rigid array property shall be represented by null elements. Any array property that uses this style shall indicate the rigid style in the long description of its schema definition.</td>
</tr>
</tbody>
</table>
Services may pad an array property with null elements at the end of the sequence to indicate the array size to clients. This is useful for small fixed length arrays, and for variable or rigid arrays with a restrictive maximum size. Services should not pad array properties if the maximum array size is not restrictive. For example, an array property typically populated with two elements, that a service limits to a maximum of 16 elements, should not pad the array with 14 null elements.

### 9.4.8. Structured properties

Structured properties are JSON objects within a response body.

Some structured properties inherit from the `Resource.v1_0_0.ReferenceableMember` definition. Structured properties that follow this definition shall contain the `MemberId` and `resource identifier` properties.

Because the definition of structured properties can evolve over time, clients need to be aware of the inheritance model that the different structured property definitions use.

For example, the `Location` definition in the `Resource` schema has gone through several iterations since the `Resource.v1_1_0` namespace was introduced, and each iteration inherits from the earlier version so that existing references in other schemas can leverage the additions.

Structured property references need to be resolved for both local and external references.

A local reference is a resource that has a structured property in its own schema, such as `ProcessorSummary` in the `ComputerSystem` resource. In these cases, the `type` property for the resource is the starting point for resolving the structured property definition.

To find the latest applicable version, clients can step the `version of the resource` backwards.

For example, if a service returns `#ComputerSystem.v1_4_0.ComputerSystem` as the resource type, a client can step backwards from `ComputerSystem.v1_4_0`, to `ComputerSystem.v1_3_0`, to `ComputerSystem.v1_2_0`, and so on, until it finds the `ProcessorSummary` structured property definition.

An external reference is a resource that has a property that references a definition found in a different schema, such as `Location` in the `Chassis` resource.

In these cases, clients can use the latest version of the external schema file as a starting point to resolve the structured property definition.

For example, if the latest version of the `Resource` schema is `1.6.0`, a client can go backward from `Resource.v1_6_0`, to `Resource.v1_5_0`, to `Resource.v1_4_0`, and so on, until it finds the `Location` structured property definition.
9.4.9. Message object

A message object provides additional information about an object, property, or error response.

A message object is a JSON object with the following properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Type</th>
<th>Required</th>
<th>Defines</th>
</tr>
</thead>
<tbody>
<tr>
<td>MessageId</td>
<td>String</td>
<td>Yes</td>
<td>The error or message. Do not confuse this value with the HTTP status code. Clients can use this code to access a detailed message from a Message Registry.</td>
</tr>
<tr>
<td>Message</td>
<td>String</td>
<td>No</td>
<td>The human-readable error message that indicates the semantics associated with the error. This shall be the complete message, and not rely on substitution variables.</td>
</tr>
<tr>
<td>RelatedProperties</td>
<td>An array of JSON pointers</td>
<td>No</td>
<td>The properties in a JSON payload that the message describes.</td>
</tr>
<tr>
<td>MessageArgs</td>
<td>An array of strings</td>
<td>No</td>
<td>The substitution parameter values for the message. If the parameterized message defines a MessageId, the service shall include the MessageArgs in the response.</td>
</tr>
<tr>
<td>Severity</td>
<td>String</td>
<td>No</td>
<td>The severity of the error.</td>
</tr>
<tr>
<td>Resolution</td>
<td>String</td>
<td>No</td>
<td>The recommended actions to take to resolve the error.</td>
</tr>
</tbody>
</table>

Each instance of a message object shall contain at least a MessageId, together with any applicable MessageArgs, or a Message property that defines the complete human-readable error message.

MessageIds identify specific messages that a Message Registry defines.

The MessageId property value shall be in the format:

<RegistryName>.MajorVersion.MinorVersion.MessageKey

where
<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RegistryName</td>
<td>Name of the registry. The registry name shall be Pascal-cased.</td>
</tr>
<tr>
<td>MajorVersion</td>
<td>Non-negative integer. The major version of the registry.</td>
</tr>
<tr>
<td>MinorVersion</td>
<td>Non-negative integer. The minor version of the registry.</td>
</tr>
<tr>
<td>MessageKey</td>
<td>Human-readable key into the registry. The message key shall be Pascal-cased and shall not include spaces, periods, or special characters.</td>
</tr>
</tbody>
</table>

To search the Message Registry for a message, the client can use the MessageId.

The Message Registry approach has advantages for internationalization because the registry can be translated easily, and is lightweight for implementations because large strings need not be included with the implementation.

The use of Base.1.0.GeneralError as a MessageId in ExtendedInfo is discouraged. If no better message exists or the ExtendedInfo array contains multiple messages, use Base.1.0.GeneralError only in the code property of the error object.

When an implementation uses Base.1.0.GeneralError in ExtendedInfo, the implementation should include a Resolution property with this error to indicate how to resolve the problem.

9.5. Properties

Every property included in a Redfish response payload shall be defined in the schema for that resource. The following attributes apply to all property definitions:

- Property names in the request and response payload shall match the casing of the Name attribute value in the defining schema.
- Required properties shall always be returned in a response.
- Properties not returned from a GET operation indicate that the property is not supported by the implementation, or by that particular resource instance. Differences in underlying product support or configuration will vary among resource instances, and therefore the properties returned by each instance will vary accordingly.
- If an implementation supports a property, it shall always provide a value for that property. If a value is unknown at the time of the operation, due to an internal error, the current resource state, or inaccessibility of the data, then the value of null is an acceptable value if supported by the schema definition.
- Resource instances should omit properties if the underlying product or service does not provide the function described by the property. For example, a chassis resource instance might not provide a serial number, and therefore should omit the SerialNumber property, while other chassis resource instances that have a serial number can provide this property.
A service may implement a writable property as read-only.

This clause also contains a set of common properties across all Redfish resources. The property names in this clause shall not be used for any other purpose.

9.5.1. Resource identifier (@odata.id) property

Registry resources in a response may include an @odata.id property. All other resources in a response shall include an @odata.id property. The value of the identifier property shall be the resource URI.

9.5.2. Resource type (@odata.type) property

All resources in a response shall include an @odata.type type property. To support generic OData clients, all structured properties in a response should include an @odata.type type property. The value shall be a URL fragment that specifies the type of the resource and shall be in the format:

#Namespace.TypeName

where

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Namespace</td>
<td>Full namespace name of the Redfish schema that defines the type. For Redfish resources, the versioned namespace name.</td>
</tr>
<tr>
<td>TypeName</td>
<td>Name of the resource type.</td>
</tr>
</tbody>
</table>

An example of a resource type value is #ComputerSystem.v1_0_0.ComputerSystem, where ComputerSystem.v1_0_0 denotes the version 1.0.0 namespace of ComputerSystem, and the type itself is ComputerSystem.

9.5.3. Resource ETag (@odataetag) property

ETags enable clients to conditionally retrieve or update a resource. Resources should include an @odata.etag property. For a resource, the value shall be the ETag.

9.5.4. Resource context (@odata.context) property

Responses for a single resource may contain an @odata.context property that describes the source of the payload.

If the @odata.context property is present, it shall be the context URL that describes the resource, according to OData-Protocol.
The context URL for a resource should be in the format:

/redfish/v1/$metadata#ResourceType

where

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResourceType</td>
<td>Fully qualified name of the unversioned resource type. Redfish resource definitions concatenate the resource type namespace with a period (.) followed by the resource type.</td>
</tr>
</tbody>
</table>

For example, the following context URL specifies that the results show a single ComputerSystem resource:

```json
{
    "@odata.context": "/redfish/v1/$metadata#ComputerSystem.ComputerSystem",
    ...
}
```

The context URL for a resource may be in one of the other formats that OData-Protocol specifies.

### 9.5.5. Id

The Id property of a resource uniquely identifies the resource within the resource collection that contains it. The value of Id shall be unique across a resource collection. The Id property shall follow the definition for Id in the Resource schema.

### 9.5.6. Name

The Name property conveys a human-readable moniker for a resource. The type of the Name property shall be string. The value of Name is NOT required to be unique across resource instances within a resource collection. The Name property shall follow the definition for Name in the Resource schema.

### 9.5.7. Description

The Description property conveys a human-readable description of the resource. The Description property shall follow the definition for Description in the Resource schema.

### 9.5.8. MemberId

The MemberId property uniquely identifies an element within an array, where a reference property can
reference the element. The MemberId value shall be unique across the array. The MemberId property shall follow the definition for MemberId in the Resource schema.

9.5.9. Count (Members@odata.count) property

The count property defines the total number of resource, or members, that are available in a resource collection. The count property shall be named Members@odata.count and its value shall be the total number of members available in the resource collection. The $top or $skip query parameters shall not affect this count.

9.5.10. Members

The Members property of a resource collection identifies the members of the collection. The Members property is required and shall be returned in the response for any resource collection. The Members property shall be an array of JSON objects named Members. The Members property shall not be null. Empty collections shall be an empty JSON array.

9.5.11. Next link (Members@odata.nextLink) property

The value of the Next Link property shall be an opaque URL to a resource, with the same @odata.type, which contains the next set of partial members from the original operation. The Next Link property shall only be present if the number of members in the resource collection is greater than the number of members returned, and if the payload does not represent the end of the requested resource collection.

The Members@odata.count property value is the total number of resources available if the client enumerates all pages of the resource collection.

9.5.12. Links

The Links property represents the hyperlinks associated with the resource, as defined by that resource’s schema definition. All associated reference properties defined for a resource shall be nested under the links property. All directly (subordinate) referenced properties defined for a resource shall be in the root of the resource.

The links property shall be named Links and contain a property for each related resource.

To navigate vendor-specific hyperlinks, the Links property shall also include an Oem property.

9.5.12.1. Reference to a related resource

A reference to a single resource is a JSON object that contains a single resource identifier property. The name of this reference is the name of the relationship. The value of this reference is the URI of the referenced resource.
9.5.12.2. References to multiple related resources

A reference to a set of zero or more related resources is an array of JSON objects. The name of this reference is the name of the relationship. Each element of the array is a JSON object that contains a resource identifier property with the value of the URI of the referenced resource.

9.5.13. Actions

The Actions property contains the actions supported by a resource.

9.5.13.1. Action representation

Each supported action is represented as a property nested under Actions. The unique name that identifies the action is used to construct the property name.

This property name shall be in the format:

`#ResourceType.ActionName`

where
<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResourceType</td>
<td>Resource where the action is defined.</td>
</tr>
<tr>
<td>ActionName</td>
<td>Name of the action.</td>
</tr>
</tbody>
</table>

The client may use this fragment to identify the action definition in the referenced Redfish schema document.

The property for the action is a JSON object and contains the following properties:

- The `target` property shall be present, and defines the relative or absolute URL to invoke the action.
- The `title` property may be present, and defines the action's name.

The OData JSON Format Specification defines the `target` and `title` properties.

To specify the list of supported values for a parameter, the service may include the `@Redfish.AllowableValues` annotation.

For example, the following property defines the Reset action for a ComputerSystem:

```json
{
  "#ComputerSystem.Reset": {
    "target": "/redfish/v1/Systems/1/Actions/ComputerSystem.Reset",
    "title": "Computer System Reset",
    "ResetType@Redfish.AllowableValues": [
      "On",
      "ForceOff",
      "GracefulRestart",
      "GracefulShutdown",
      "ForceRestart",
      "Nmi",
      "ForceOn",
      "PushPowerButton"
    ]
  },
  ...
}
```

Given this, the client could invoke a POST request to `/redfish/v1/Systems/1/Actions/ComputerSystem.Reset` with the following body:
POST /redfish/v1/Systems/1/Actions/ComputerSystem.Reset HTTP/1.1
Content-Type: application/json;charset=utf-8
Content-Length: <computed length>
OData-Version: 4.0

{
  "ResetType": "On"
}

The resource may provide a separate @Redfish.ActionInfo resource to describe the parameters and values that a particular instance or implementation supports. Use the @Redfish.ActionInfo annotation to specify the ActionInfo resource, which contains a URI to the @Redfish.ActionInfo resource for the action. For details, see the Action info annotation clause.

9.5.13.2. Action responses

Response payloads for actions may contain a JSON body that is described by the schema definition for the action. See the Schema definition languages clause for the representation of these definitions. Actions that do not define a response body may provide an error response in the response payload.

9.5.14. Oem

The Oem property is used for Resource extensibility-defined OEM extensions.

9.5.15. Status

The Status property represents the status of a resource. The Status property shall follow the definition for Status in the Resource schema.

By having a common representation of status, clients can depend on consistent semantics. The Status property is capable of indicating the current state, health of the resource, and the health of subordinate resources.

9.6. Resource, schema, property, and URI naming conventions

The Redfish interface is intended to be easily readable and intuitive. Thus, consistency helps the consumer who is unfamiliar with a newly discovered property understand its use. While this is no substitute for the normative information in the Redfish Specification and Redfish schema, the following rules help with readability and client usage. In general, names in Redfish are designed and intended to be human-readable and convey the meaning of the name, in context, without the need to consult schema definitions or other documentation.

Standard Redfish resources defined and published in the repository, or those created by others and...
republished, shall follow a set of naming conventions. These conventions are intended to ensure
consistent naming and eliminate naming collisions. The resource name is used to construct the type
property and the schema file name.

Standard Redfish properties follow similar naming conventions, and should use a common definition
when defined in multiple schemas across the Redfish data model. This consistency enables code re-use
across resources and increases interoperability. New resource definitions should leverage existing
property definitions whenever possible.

The naming rules for schemas, properties, and enumerations are as follows:

- Names shall be Pascal-cased. The first letter of each word in a name shall be uppercase and
  spaces between words shall be removed. For example, ComputerSystem, PowerState, and
  SerialNumber.
- Names shall not contain spaces or underscore characters. Names should not contain any
  special characters that violate naming rules for supported schema description languages or
  programming languages.
- Both characters should be capitalized for two-character acronyms. For example, IPAddress or
  RemoteIP.
- Names constructed from a single acronym or mixed-case name, such as LDAP, PCIe, or SNMP,
  should use the typical capitalization for that name.
- Names incorporating acronyms with three or more characters should follow the capitalization
  used in related names for consistency. For example, EnableSNMPv1 and EnableSNMPv2
  follow the pattern used for SNMP.
- Pascal-casing may be used for acronyms longer than two characters to improve readability,
  especially when two or more acronyms appear together in a name, which should be avoided.
- Enumeration names should start with a letter and be followed by letters or numbers, in order to
  conform to schema description language requirements. Underscore characters may be used to
  replace other special characters, or to significantly improve readability, but this usage is
discouraged.
- Enumeration names should prioritize readability as they may appear unmodified on user
  interfaces, whereas property or schema names should follow conventions and strive for
  consistency.

Exceptions are allowed for the following cases:

- Well-known technology abbreviations, acronyms, or product names should follow their defined
  capitalization. Examples include iSCSI, iSCSI Target, and iLO.
- OEM appears as Oem in schema and property names either alone or as a portion of a name, but
  should be OEM when used alone as an enumeration value.

For properties that have units or other special meaning, append a unit identifier to the name. Examples
include:
• Bandwidth (Mbps). For example, PortSpeedMbps.
• CPU speed (Mhz). For example, ProcessorSpeedMhz.
• Memory size (MB). For example, MemoryMB.
• Counts of items (Count). For example, ProcessorCount or FanCount.
• The state of a resource (State). For example, PowerState.
• State values where work is in process. For example, Applying or ClearingLogic.

In addition, the following rules apply to Redfish schema-defined URIs:

• URI segments should generally follow the naming rules, and follow the name of the Redfish schema that defines the resource located at each segment.
• URI segments for resource collections should use the plural form of the resource collection schema name, with the Collection term omitted. For example, Processors for a ProcessorCollection.
• For resources that contain hyperlinks to more than one resource or resource collection of the same schema type, the URI segments should follow the name of the property that provides the hyperlink, for clarity.
• If a hyperlink to a subordinate resource is not found at the root of the resource, the URI segments should contain the property path. For example, for the Certificates hyperlink found in ManagerNetworkProtocol within the HTTPS object, HTTPS should be one of the URI segments.

9.7. Resource extensibility

In the context of this clause, the OEM term refers to any company, manufacturer, or organization that provides or defines an extension to the DMTF-published schema and functionality for Redfish. All Redfish-specified resources include an empty structured Oem property. Its value can encapsulate one or more OEM-specified structured properties. This predefined placeholder can contain OEM-specific property definitions.

9.7.1. OEM property format and content

Each property contained within the Oem property shall be a JSON object. The name of the object (property) shall uniquely identify the OEM or organization that defines the properties contained by that object. This is described in more detail in the following clause.

The OEM-specified object shall include a type property that provides the location of the schema and the type definition for the property within that schema if the OEM-specified object:

• Is not contained in an array of objects.
• Is contained in the first object within an array of objects.
• In subsequent array members containing an OEM-specified object, whose type is different than the first array member.
The **Oem** property can simultaneously hold multiple OEM-specified objects, including objects for more than one company or organization.

The definition of any other properties that are contained within the OEM-specific object, along with the functional specifications, validation, or other requirements for that content is OEM-specific and outside the scope of this specification. While there are no Redfish-specified limits on the size or complexity of the OEM-specified elements within an OEM-specified JSON object, it is intended that OEM properties typically be used for only a small number of simple properties that augment the Redfish resource. If a large number of objects or a large quantity of data compared to the size of the Redfish resource is to be supported, the OEM should consider having the OEM-specified object point to a separate resource for their extensions.

### 9.7.2. OEM property naming

The OEM-specified objects within the **Oem** property are named by using a unique OEM identifier for the top of the namespace under which the property is defined. There are two specified forms for the identifier. The identifier shall be either an ICANN-recognized domain name (including the top-level domain suffix), with all dot (.) separators replaced with underscores (_), or an IANA-assigned Enterprise Number prefixed with "EID:.

**DEPRECATED:** The identifier shall be either an ICANN-recognized domain name including the top-level domain suffix, or an IANA-assigned Enterprise Number prefixed with "EID:.

Organizations that use .com domain names may omit the .com suffix. For example, Contoso.com would use Contoso instead of Contoso_com, but Contoso.org would use Contoso_org. The domain name portion of an OEM identifier shall be considered to be case independent. That is, the text Contoso_biz, contoso_BIZ, conTOso_biZ, and so on all identify the same OEM and top-level namespace.

The OEM identifier portion of the property name may be followed by an underscore (_) and any additional string to enable further creation of namespaces of OEM-specified objects as desired by the OEM. For example, Contoso xxxx or EID_412 xxxx. The form and meaning of any text that follows the trailing underscore is completely OEM-specific. OEM-specified extension suffixes may be case sensitive, depending on the OEM. Generic client software should treat such extensions, if present, as opaque and not try to parse nor interpret the content.

This suffix could be used in many ways, depending on OEM need. For example, the Contoso company may have a *Research* suborganization, in which case the OEM-specified property name might be extended to _ContosoResearch. Alternatively, it can identify a namespace for a functional area, geography, subsidiary, and so on.

The OEM identifier portion of the name typically identifies the company or organization that created and maintains the schema for the property. However, this is not a requirement. The identifier is only required to uniquely identify the party that is the top-level manager of a namespace to prevent collisions between OEM property definitions from different vendors or organizations. Consequently, the organization for the
top of the namespace may be different than the organization that provides the definition of the OEM-specified property. For example, Contoso may allow one of their customers, such as CustomerA, to extend a Contoso product with certain CustomerA proprietary properties. In this case, although Contoso allocated the name `Contoso_customers_CustomerA`, it could be CustomerA that defines the content and functionality under that namespace. In all cases, OEM identifiers should not be used except with permission or as specified by the identified company or organization.

### 9.7.3. OEM resource naming and URIs

Companies, OEMs, and other organizations can define additional resources and link to them from an `Oem` property found in a standard Redfish resource. To avoid naming collisions with current or future standard Redfish schema files, the defining organization's name should be prepended to the resource name. For example, `ContosoDrive` would not conflict with a `Drive` resource or another OEM's drive-related resource.

To avoid URI collisions with other OEM resources and future Redfish standard resources, the URIs for OEM resources within the Redfish resource tree shall be in the form:

```
BaseUri/Oem/OemName/ResourceName
```

where

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BaseUri</td>
<td>URI segment of the standard Redfish resource starting with <code>/redfish/</code> where the <code>Oem</code> property is used. For example, <code>/redfish/v1/Systems/3A238944T523</code>.</td>
</tr>
<tr>
<td>OemName</td>
<td>Name of the OEM, that follows the same naming as defined in the <code>Oem</code> property format and content clause.</td>
</tr>
<tr>
<td>ResourceName</td>
<td>Name of the OEM-defined resource.</td>
</tr>
</tbody>
</table>

For example, if Contoso defined a new `ContosoAccountServiceMetrics` resource to be linked through the `Oem` property at the `/redfish/v1/AccountService` URI, the OEM resource has the `/redfish/v1/AccountService/Oem/Contoso/AccountServiceMetrics` URI.

### 9.7.4. OEM property examples

The following fragment presents some examples of naming and use of the `Oem` property as it might appear when accessing a resource. The example shows that the OEM identifiers can be of different forms, that OEM-specified content can be simple or complex, and that the format and usage of extensions of the OEM identifier is OEM-specific.
9.7.5. OEM actions

OEM-specific actions appear in the JSON payload as properties of the `Oem` object, nested under an `Actions` property.

The name of the property that represents the action, which shall follow the form:

`#Namespace.Action`

where

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>Namespace</code></td>
<td>Namespace.</td>
</tr>
<tr>
<td><code>Action</code></td>
<td>Action.</td>
</tr>
</tbody>
</table>
The URI of the OEM action in the `target` property shall be in the form:

`ResourceURI/Actions/Oem/Namespace.Action`

where

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResourceUri</td>
<td>URI of the resource that supports invoking the action. For example, <code>/redfish/v1/Systems/1/</code>.</td>
</tr>
<tr>
<td>Actions</td>
<td>Name of the property containing the actions for a resource.</td>
</tr>
<tr>
<td>Oem</td>
<td>Name of the OEM property within the <code>Actions</code> property.</td>
</tr>
<tr>
<td>Namespace.Action</td>
<td>Namespace followed by the action. For example, <code>Contoso.Ping</code>.</td>
</tr>
</tbody>
</table>

9.8. Payload annotations

`Resources`, `objects within a resource`, and `properties` may include additional annotations as properties with the name, in the format:

`[PropertyName]@Namespace.TernName`

where

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PropertyName</td>
<td>Name of the property to annotate. If absent, the annotation applies to the entire JSON object, which may be an entire resource.</td>
</tr>
<tr>
<td>Namespace</td>
<td>Namespace that defines the annotation term.</td>
</tr>
<tr>
<td>Variable</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>-------------</td>
</tr>
<tr>
<td>TermName</td>
<td>Annotation term to apply to the resource or property of the resource.</td>
</tr>
</tbody>
</table>

Services shall limit the annotation usage to the odata, Redfish, and Message namespaces. The OData JSON Format Specification defines the odata namespace. The Redfish namespace is an alias for the RedfishExtensions.v1_0_0 namespace.

The client can get the definition of the annotation from the OData metadata document, the HTTP Link header, or may ignore the annotation entirely, but should not fail reading the resource due to unrecognized annotations, including new annotations that the Redfish namespace defines.

### 9.8.1. Allowable values

To specify the list of allowable values for a property or action parameter, services may use the @Redfish.AllowableValues annotation for properties or action parameters.

To specify the set of allowable values, include a property with the name of the property or action parameter, followed by @Redfish.AllowableValues. The property value is a JSON array of strings that define the allowable values for the property or action parameter.

### 9.8.2. Extended information

The following clauses describe the methods of providing extended information:

- [Extended object information](#)
- [Extended property information](#)

#### 9.8.2.1. Extended object information

To specify object-level status information, services may annotate a JSON object with the @Message.ExtendedInfo annotation.

```json
{
  "@odata.id": "/redfish/v1/Managers/1/SerialInterfaces/1",
  "@odata.type": "#SerialInterface.v1_0_0.SerialInterface",
  "Name": "Managed Serial Interface 1",
  "Description": "Management for Serial Interface",
  "Status": {
    "State": "Enabled",
    "Health": "OK"
  },
  "InterfaceEnabled": true
}
```
The property contains an array of message objects.

9.8.2.2. Extended property information

Services may use @Message.ExtendedInfo, prepended with the name of the property to annotate an individual property in a JSON object with extended information:
"FlowControl": "None",
"ConnectorType": "RJ45",
"PinOut": "Cyclades",
"PinOut@Message.ExtendedInfo": [
    {
        "MessageId": "Base.1.0.PropertyValueNotInList",
        "Message": "The value Contoso for the property PinOut is not in the list of acceptable values."
    }
]

9.8.3. Action info annotation

The action info annotation conveys the parameter requirements and allowable values on parameters for actions. This is done using @Redfish.ActionInfo term within the action representation. This term contains a URI to the ActionInfo resource.

Example #ComputerSystem.Reset action with the @Redfish.ActionInfo annotation and resource:

```
{
    "Actions": {
        "#ComputerSystem.Reset": {
            "target": "/redfish/v1/Systems/1/Actions/ComputerSystem.Reset",
            "@Redfish.ActionInfo": "/redfish/v1/Systems/1/ResetActionInfo"
        }
    },
    ...
}
```

The ResetActionInfo resource contains a more detailed description of the parameters and the supported values. This resource follows the ActionInfo schema definition.

```
{
    "@odata.id": "/redfish/v1/Systems/1/ResetActionInfo",
    "@odata.type": "#ActionInfo.v1_0_0.ActionInfo",
    "Id": "ResetActionInfo",
    "Name": "Reset Action Info",
    "Parameters": [
```
9.8.4. Settings and settings apply time annotations

See the Settings resource clause.

9.8.5. Operation apply time and operation apply time support annotations

See the Operation apply time clause.

9.8.6. Maintenance window annotation

The settings apply time and operation apply time annotations allow for an operation to be performed during a maintenance window. The @Redfish.MaintenanceWindow term at the root of a resource configures the start time and duration of a maintenance window for a resource.

The following example body for the /redfish/v1/Systems/1 resource configures the maintenance window to start at 2017-05-03T23:12:37-05:00 and last for 600 seconds.

```json
{
    "@odata.id": "/redfish/v1/Systems/1",
    "@odata.type": "#ComputerSystem.v1_5_0.ComputerSystem",
    "@Redfish.MaintenanceWindow": {
        "MaintenanceWindowStartTime": "2017-05-03T23:12:37-05:00",
        "MaintenanceWindowDurationInSeconds": 600
    }
}
```
9.8.7. Collection capabilities annotation

Resource collections may contain a collection capabilities annotation. The @Redfish.CollectionCapabilities term at the root of a resource collection shows what properties a client is allowed to use in a POST request for creating a new resource.

The following ComputerSystemCollection example body contains the collection capabilities annotation. The UseCase property contains the ComputerSystemComposition value, and the CapabilitiesObject property contains the /redfish/v1/Systems/Capabilities value. The resource at /redfish/v1/Systems/Capabilities describes the POST request format for creating a ComputerSystem resource for compositions.

```json
{
  "@odata.id": "/redfish/v1/Systems",
  "@odata.type": "#ComputerSystemCollection.ComputerSystemCollection",
  "Name": "Computer System Collection",
  "Members@odata.count": 0,
  "Members": [],
  "@Redfish.CollectionCapabilities": {
    "@odata.type": "#CollectionCapabilities.v1_1_0.CollectionCapabilities",
    "Capabilities": [
      {
        "CapabilitiesObject": {
          "@odata.id": "/redfish/v1/Systems/Capabilities"
        },
        "UseCase": "ComputerSystemComposition",
        "Links": {
          "TargetCollection": {
            "@odata.id": "/redfish/v1/Systems"
          }
        }
      }
    ]
  }
}
```

The CapabilitiesObject resource follows the same schema for the resource that the resource collection contains. It contains annotations to show which properties the client is allowed to use in the POST request body. The annotations describe which properties are required, optional, or if other rules are associated with the properties.

<table>
<thead>
<tr>
<th>Annotation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><a href="mailto:PropertyName@Redfish.RequiredOnCreate">PropertyName@Redfish.RequiredOnCreate</a></td>
<td>Required in the POST request body.</td>
</tr>
</tbody>
</table>
### Annotation

<table>
<thead>
<tr>
<th>Annotation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>PropertyName@Redfish.OptionalOnCreate</code></td>
<td>Not required in the POST request body.</td>
</tr>
<tr>
<td><code>PropertyName@Redfish.SetOnlyOnCreate</code></td>
<td>Cannot be modified after the resource is created.</td>
</tr>
<tr>
<td><code>PropertyName@Redfish.UpdatableAfterCreate</code></td>
<td>Can be modified after the resource is created.</td>
</tr>
<tr>
<td><code>PropertyName@Redfish.AllowableValues</code></td>
<td>Can be set to any of the listed values.</td>
</tr>
<tr>
<td><code>@Redfish.RequestedCountRequired</code></td>
<td>Required in the POST request body for the corresponding object to indicate the number of requested object instances. Used for composition requests.</td>
</tr>
<tr>
<td><code>@Redfish.ResourceBlockLimits</code></td>
<td>Indicates restrictions regarding quantities of ResourceBlock resources of a given type in the POST request body. Used for composition requests.</td>
</tr>
</tbody>
</table>

**Example** CapabilitiesObject resource:

```json
{
  "@odata.id": "/redfish/v1/Systems/Capabilities",
  "@odata.type": ":$ComputerSystem.v1_8_0.ComputerSystem",
  "Id": "Capabilities",
  "Name": "Capabilities for the system collection",
  "Name@Redfish.RequiredOnCreate": true,
  "Name@Redfish.SetOnlyOnCreate": true,
  "Description@Redfish.OptionalOnCreate": true,
  "Description@Redfish.SetOnlyOnCreate": true,
  "HostName@Redfish.OptionalOnCreate": true,
  "HostName@Redfish.UpdatableAfterCreate": true,
  "Links@Redfish.RequiredOnCreate": true,
  "Links": {
    "ResourceBlocks@Redfish.RequiredOnCreate": true,
    "ResourceBlocks@Redfish.UpdatableAfterCreate": true
  }
}
```
9.8.8. Requested count and allow over-provisioning annotations

Clients use the requested count and allow over-provisioning annotations in composition requests to express the quantity of a type of resource to allocate:

<table>
<thead>
<tr>
<th>Annotation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>@Redfish.RequestedCount</td>
<td>Number of requested resources.</td>
</tr>
<tr>
<td>@Redfish.AllowOverprovisioning</td>
<td>Boolean. If true, the service may provision more resources than the @Redfish.RequestedCount annotation requests. Default is false.</td>
</tr>
</tbody>
</table>

Example client request for four or more Processor resources:

```
{
    "Processors": {
        "Members": [
            {
                "@Redfish.RequestedCount": 4,
                "@Redfish.AllowOverprovisioning": true
            }
        ]
    },
    ...
}
```

9.8.9. Zone affinity annotation

The zone affinity annotation is used by clients in composition requests to indicate the components for the composition come from the specified Resource Zone. The @Redfish.ZoneAffinity term in the request body contains the value of the Id property of the requested Resource Zone.

Example client request for components to be allocated from the Resource Zone with the Id property DSP0266.
9.8.10. Supported certificates annotation

Resource collections of type CertificateCollection should contain a supported certificates annotation. The @Redfish.SupportedCertificates term at the root of a resource collection shows the different certificate formats allowed in the resource collection.

Example CertificateCollection that only supports PEM style certificates:

```json
{
  "@odata.id": "/redfish/v1/Managers/BMC/NetworkProtocol/HTTPS/Certificates",
  "@odata.type": "#CertificateCollection.CertificateCollection",
  "Name": "Certificate collection",
  "Members@odata.count": 1,
  "Members": [
    {
      "@odata.id": "/redfish/v1/Managers/BMC/NetworkProtocol/HTTPS/Certificates/1"
    }
  ],
  "@Redfish.SupportedCertificates": ["PEM"]
}
```

9.8.11. Deprecated annotation

Services may annotate properties with @Redfish.Deprecated if the schema definition has the property marked as deprecated.

Example deprecated property:

```json
{
  "VendorID": "GxABCD",
  "VendorID@Redfish.Deprecated": "This property has been deprecated in favor of"
}
```
9.9. Settings resource

A settings resource represents the future intended state of a resource. Some resources have properties that can be updated and the updates take place immediately; however, some properties need to be updated at a certain point in time, such as a system reset. While the resource represents the current state, the settings resource represents the future intended state. The service represents properties of a resource that can only be updated at a certain point in time using a @Redfish.Settings payload annotation. The settings annotation contains a link to a subordinate resource with the same schema definition. The properties within the settings resource contain the properties that are updated at a certain point in time.

For resources that support a future state and configuration, the response shall contain a property with the @Redfish.Settings annotation. When a settings annotation is used, the following conditions shall apply:

- The settings resource linked to current resource with the @Redfish.Settings annotation shall be of the same schema definition.
- The settings resource should be a subset of properties that can be updated.
- The settings resource shall not contain the @Redfish.Settings annotation.
- The settings resource may contain the @Redfish.SettingsApplyTime annotation.

The settings resource includes several properties to help clients monitor when the resource is consumed by the service and determine the results of applying the values, which may or may not have been successful.

- The Messages property is a collection of messages that represent the results of the last time the values of the settings resource were applied.
- The ETag property contains the ETag of the settings resource that was last applied.
- The Time property indicates the time when the settings resource was last applied.

The following resource example body supports a settings resource. A client can use the SettingsObject property to locate the URI of the settings resource.

```json
{
   "@Redfish.Settings": {
      "@odata.type": "#Settings.v1_0_0.Settings",
      "SettingsObject": {
```
When a client updates the settings resource, it may include the `@Redfish.SettingsApplyTime` annotation in the request to indicate when to apply the settings.

- If a service enables a client to indicate when to apply settings, the settings resource shall contain a property with the `@Redfish.SettingsApplyTime` annotation.
- Only settings resources shall contain the `@Redfish.SettingsApplyTime` annotation.

In the following example request, the client indicates that the settings resource values are applied either on reset or during the specified maintenance window:

```json
{
  "@Redfish.SettingsApplyTime": {
    "@odata.type": "#Settings.v1_1_0.PreferredApplyTime",
    "ApplyTime": "OnReset",
    "MaintenanceWindowStartTime": "2017-05-03T23:12:37-05:00",
    "MaintenanceWindowDurationInSeconds": 600
  },
  ...
}
```

### 9.10. Special resource situations

#### 9.10.1. Overview

Resources need to exhibit common semantic behavior whenever possible. This can be difficult in some situations discussed in this clause.
9.10.2. Absent resources

Resources may be absent or their state unknown at the time a client requests information about that resource. For resources that represent removable or optional components, absence provides useful information to clients because it indicates a capability, such as an empty PCIe slot, DIMM socket, or drive bay, that would not be apparent if the resource simply did not exist.

This also applies to resources that represent a limited number of items or unconfigured capabilities within an implementation, but this usage should be applied sparingly and should not apply to resources limited in quantity due to arbitrary limits. For example, an implementation that limits SoftwareInventory to a maximum of 20 items should not populate 18 absent resources when only two items are present.

For resources that provide useful data in an absent state and where the URI is expected to remain constant, such as when a DIMM is removed from a memory socket, the resource should exist and should return a value of Absent for the State property in the Status object.

In this circumstance, any required properties for which there is no known value shall be represented as null. Properties whose support is based on the configuration choice or the type of component installed, and therefore unknown while in the absent state, should not be returned. Likewise, subordinate resources for an absent resource should not be populated until their support can be determined. For example, the Power and Thermal resources under a Chassis resource should not exist for an absent Chassis.

Client software should be aware that when absent resources are later populated, the updated resource may represent a different configuration or physical item, and previous data, including read-only properties, obtained from that resource may be invalid. For example, the Memory resource shows details about an single DIMM socket and the installed DIMM. When that DIMM is removed, the Memory resource remains as an absent resource to indicate the empty DIMM socket. Later, a new DIMM is installed in that socket, and the Memory resource represents data about this new DIMM, which could have completely different characteristics.

9.11. Registries

Registry resources assist the client in interpreting Redfish resources beyond the Redfish schema definitions. To get more information about a resource, event, message, or other item, use an identifier to search registries. This information can include other properties, property restrictions, and the like. Registries are themselves resources.

Redfish defines the following types of registries:
### Registry

<table>
<thead>
<tr>
<th>Registry</th>
<th>Description</th>
<th>See</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIOS</td>
<td>Determines the semantics of each property in a BIOS or BIOS settings resource. Because BIOS information can vary from platform to platform, Redfish cannot define a fixed schema for these values. This registry contains both property descriptions and other information, such as data type, allowable values, and user menu information.</td>
<td></td>
</tr>
<tr>
<td>Message</td>
<td>Constructs a message from a MessageId and other message information to present to an end user. The messages in these registries appear in both eventing and error responses to operations. This registry is the most common type of registry.</td>
<td>• Error responses • Eventing</td>
</tr>
<tr>
<td>Privilege</td>
<td>Maps the resources in a Redfish Service to the privileges that can complete specified operations against those resources. A client can use this information to: • Determine which roles should have specific privileges. • Map accounts to those roles so that the accounts can complete operations on Redfish resources.</td>
<td>Privilege model</td>
</tr>
</tbody>
</table>

### 9.12. Schema annotations

The schema definitions of the data model use schema annotations to provide additional documentation for developers. This clause describes the different types of schema annotations that the Redfish data model uses. For information about how each of the annotations are implemented in their respective schema languages, see the Schema definition languages clause.

#### 9.12.1. Description annotation

The description annotation can be applied to any type, property, action, or parameter to provide a description of Redfish schema elements suitable for end users or user interface help text.
A description annotation shall be included on the following schema definitions:

- Redfish types
- Properties
- Reference properties
- Enumeration values
- Resources and resource collections
- Structured types

### 9.12.2. Long description annotation

The long description annotation can be applied to any type, property, action, or parameter to provide a formal, normative specification of the schema element.

When the long descriptions in the Redfish schema contain normative language, the service shall be required to conform with the statement.

A long description annotation shall be included on the following schema definitions:

- Redfish types
- Properties
- Reference properties
- Resources and resource collections
- Structured types

### 9.12.3. Resource capabilities annotation

The resource capabilities annotation can be applied to resources and resource collections to express the different type of HTTP operations a client can invoke on the given resource or resource collection.

- Insert capabilities indicate whether a client can perform a POST on the resource.
- Update capabilities indicate whether a client can perform a PATCH or PUT on the resource.
- Delete capabilities indicate whether a client can perform a DELETE on the resource.
- A service may implement a subset of the capabilities that are allowed on the resource or resource collection.

All schema definitions for Redfish resources and resource collections shall include resource capabilities annotations.

### 9.12.4. Resource URI patterns annotation

The resource URI patterns annotation expresses the valid URI patterns for a resource or resource collection.

The strings for the URI patterns may use { and } characters to express parameters within a given URI.
pattern. Items between the { and } characters are treated as identifiers within the URI for given instances of a Redfish resource. Clients interpret this as a string to be replaced to access a given resource. A URI pattern may contain multiple identifier terms to support multiple levels of nested resource collections. The identifier term in the URI pattern shall match the Id string property for the corresponding resource, or the MemberId string property for the corresponding object within a resource. The process for forming the strings that are concatenated to form the URI pattern are in the resource, schema, property, and URI naming conventions clause.

The following string is an example URI pattern that describes a ManagerAccount resource: /redfish/v1/AccountService/Accounts/{ManagerAccountId}

Using the previous example, {ManagerAccountId} is replaced by the Id property of the corresponding ManagerAccount resource. If the Id property for a ManagerAccount resource is John, the full URI for that resource is /redfish/v1/AccountService/Accounts/John.

The URI patterns are constructed based on the formation of the resource tree. When constructing the URI pattern for a subordinate resource, the URI pattern for the current resource is used and appended. For example, the RoleCollection resource is subordinate to AccountService. Because the URI pattern for AccountService is /redfish/v1/AccountService, the URI pattern for the RoleCollection resource is /redfish/v1/AccountService/Roles.

In some cases, the subordinate resource is found inside of a structured property of a resource. In these cases, the name of the structured property appears in the URI pattern for the subordinate resource. For example, the CertificateCollection resource is subordinate to the ManagerNetworkProtocol resource from the HTTPS property. Because the URI pattern for ManagerNetworkProtocol is /redfish/v1/Managers/{ManagerId}/NetworkProtocol, the URI pattern for the CertificateCollection resource is /redfish/v1/Managers/{ManagerId}/NetworkProtocol/HTTPS/Certificates.

All schema definitions for Redfish resources and Redfish resource collections shall be annotated with the resource URI patterns annotation.

All Redfish resources and Redfish resource collections implemented by a service shall match the URI pattern described by the resource URI patterns annotation for their given definition.

9.12.5. Additional properties annotation

The additional properties annotation specifies whether a type can contain additional properties outside of those defined in the schema. Types that do not support additional properties shall not contain properties beyond those described in the schema.
9.12.6. Permissions annotation

The permissions annotation specifies whether a client can modify the value of a property, or if the property is read-only.

A service may implement a modifiable property as read-only.

9.12.7. Required annotation

The required annotation specifies whether a service needs to support a property. Required properties shall be annotated with the required annotation. All other properties are optional.

9.12.8. Required on create annotation

The required on create annotation specifies that a property is required to be provided by the client on creation of the resource. Properties not annotated with the required on create annotation are not required to be provided by the client on a create operation.

9.12.9. Units of measure annotation

In addition to following naming conventions, properties representing units of measure shall be annotated with the units of measure annotation to specify the units of measurement for the property.

The value of the annotation shall be a string that contains the case-sensitive "(c/s)" symbol of the unit of measure as listed in the [Unified Code for Units of Measure (UCUM)](https://unitsofmeasure.org), unless the symbolic representation does not reflect common usage. For example, RPM commonly reports fan speeds in revolutions-per-minute but has no simple UCUM representation. For units with prefixes, the case-sensitive (c/s) symbol for the prefix as listed in UCUM should be prepended to the unit symbol. For example, Mebibyte (1024^2 bytes), which has the UCUM Mi prefix and By symbol, would use MiBy as the value for the annotation. For values that also include rate information, such as megabits per second, the rate unit's symbol should be appended and use a slash (/) character as a separator. For example, Mbit/s.

9.12.10. Expanded resource annotation

The expanded resource annotation can be applied to a reference property to specify that the default behavior for the service is to include the contents of the related resource or resource collection in responses. This behavior follows the same semantics of the expand query parameter with a level of 1.

Reference properties annotated with this term shall be expanded by the service, even if not requested by the client. A service may page resource collections.
**9.12.11. Owning entity annotation**

The owning entity annotation can be applied to a schema to specify the name of the entity responsible for development, publication, and maintenance of a given schema.


The deprecated annotation specifies if a property, enumeration, or other schema element has been deprecated. Schema elements marked as deprecated contain a schema version that shows when the element was deprecated, as well as text that specifies the favored approach.

Existing and new implementations may use deprecated schema elements, but they should move to the favored approach. Deprecated schema elements may be implemented in order to achieve backwards compatibility. Deprecated schema elements may be removed from the next major version of the schema.

**9.13. Versioning**

As stated previously, a resource can be an individual entity or a resource collection, which acts as a container for a set of resources.

A resource collection does not contain any version information because it defines a single Members property, and the overall collection definition never grows over time.

A resource has both unversioned and versioned definitions.

References from other resources use the unversioned definition of a resource to ensure no version dependencies exist between the definitions. The unversioned definition of a resource contains no property information about the resource.

The versioned definition of a resource contains a set of properties, actions, and other definitions associated with the given resource. The version of a resource follows the format:

\[ vX.Y.Z \]

where

<table>
<thead>
<tr>
<th>Variable</th>
<th>Type</th>
<th>Version</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( X )</td>
<td>Integer</td>
<td>Major version.</td>
<td>Backward-incompatible change.</td>
</tr>
<tr>
<td>( Y )</td>
<td>Integer</td>
<td>Minor version.</td>
<td>Minor update. Redfish introduces new functionality but does not remove any functionality. The minor version preserves compatibility with earlier minor versions. For example, a new</td>
</tr>
<tr>
<td>Variable</td>
<td>Type</td>
<td>Version</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>--------</td>
<td>---------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>2</td>
<td>Integer</td>
<td>Errata</td>
<td>Fix in an earlier version. For example, a fix to a schema annotation on a property introduces an errata version of the resource.</td>
</tr>
</tbody>
</table>


The creation of separate localized copies of Redfish schemas and registries is allowed and encouraged. Localized schema and registry files may be submitted to the DMTF for republication in the Redfish Schema Repository.

Property names, parameter names, and enumeration values in the JSON response payload are never localized but translated copies of those names may be provided as additional annotations in the localized schema for use by client applications. A separate file for each localized schema or registry shall be provided for each supported language. The English-language versions of Redfish schemas and registries shall be the normative versions, and alterations of meaning due to translation in localized versions of schemas and registries shall be forbidden.

Schemas and registries in non-English languages shall use the appropriate schema annotations to identify their language. Descriptive property, parameter, and enumeration text not translated into the specified language shall be removed from localized versions. This removal enables software and tools to combine normative and localized copies, especially for minor schema version differences.

### 10. File naming and publication

For consistency in publication and to enable programmatic access, all Redfish-related files shall follow a set of rules to construct the name of each file. The [Schema definition languages](#) clause describes the file name construction rules, while the following clauses describe the construction rules for other file types.

#### 10.1. Registry file naming

Redfish Message or Privilege Registry Files shall use the registry name to construct the file name, in this format:

```
RegistryName.MajorVersion.MinorVersion.Errata.json
```

For example, the file name of the Base Message Registry v1.0.2 is `Base.1.0.2.json`. 
10.2. Profile file naming

The document that describes a profile follows the Redfish schema file naming conventions. The file name format for profiles shall be:

ProfileName.vMajorVersion_MinorVersion_Errata.json

For example, the file name of the BasicServer profile v1.2.0 is BasicServer.v1_2_0.json. The file name shall include the profile name and version, which matches those property values within the document.

10.3. Dictionary file naming

The binary file describing a Redfish Device Enablement Dictionary follows the Redfish schema file naming conventions for the schema definition language that the dictionary is converted from. Because a single Dictionary file contains all minor revisions of the schema, only the major version appears in the file name. The file names for Dictionaries shall be formatted as:

DictionaryName_vMajorVersion.dict

For example, the file name of the Chassis dictionary v1.2.0 is Chassis_v1.dict.

10.4. Localized file naming

Localized schemas and registries shall follow the same file naming conventions as the English language versions. When multiple localized copies are present in a repository and which have the same file name, files in languages other than English shall be organized into subfolders named to match the ISO 639-1 language code for those files. English language files may be duplicated in an en subfolder for consistency.

10.5. DMTF Redfish file repository

All Redfish schemas, registries, dictionaries, and profiles published or republished by the DMTF’s Redfish Forum are available from the DMTF website http://redfish.dmtf.org/ for download. Programs may use the following durable URLs to access the repository. Programs incorporating remote repository access should implement a local cache to reduce latency, program requirements for Internet access and undue traffic burden on the DMTF website.

Organizations creating Redfish-related files such as OEM schemas, Redfish Interoperability Profiles, or Message Registries are encouraged to use the form at https://redfish.dmtf.org/redfish/portal to submit those files to the DMTF for republication in the DMTF Redfish file repository.
The files are organized on the site in the following manner:

<table>
<thead>
<tr>
<th>URL</th>
<th>Folder contents</th>
</tr>
</thead>
<tbody>
<tr>
<td>redfish.dmtf.org/schemas</td>
<td>Current (most recent minor or errata) release of each schema file in CSDL, JSON Schema, and/or OpenAPI formats.</td>
</tr>
<tr>
<td>redfish.dmtf.org/schemas/v1</td>
<td>Durable URL for programmatic access to all v1.xx schema files. Every v1.xx minor or errata release of each schema file in CSDL, JSON Schema, OpenAPI formats.</td>
</tr>
<tr>
<td>redfish.dmtf.org/schemas/v1/{code}</td>
<td>Durable URL for programmatic access to localized v1.xx schema files. Localized schemas are organized in subfolders using the two-character ISO 639-1 language code as the {code} segment.</td>
</tr>
<tr>
<td>redfish.dmtf.org/schemas/archive</td>
<td>Subfolders contain schema files specific to a particular version release.</td>
</tr>
<tr>
<td>redfish.dmtf.org/registries</td>
<td>Current (most recent minor or errata) release of each registry file.</td>
</tr>
<tr>
<td>redfish.dmtf.org/registries/v1</td>
<td>Durable URL for programmatic access to all v1.xx registry files. Every v1.xx minor or errata release of each registry file.</td>
</tr>
<tr>
<td>redfish.dmtf.org/registries/v1/{code}</td>
<td>Durable URL for programmatic access to localized v1.xx registry files. Localized schemas are organized in subfolders using the two-character ISO 639-1 language code as the {code} segment.</td>
</tr>
<tr>
<td>redfish.dmtf.org/registries/archive</td>
<td>Subfolders contain registry files specific to a particular version release.</td>
</tr>
<tr>
<td>redfish.dmtf.org/profiles</td>
<td>Current release of each Redfish Interoperability Profile (.json) file and associated documentation.</td>
</tr>
<tr>
<td>redfish.dmtf.org/profiles/v1</td>
<td>Durable URL for programmatic access to all v1.xx Redfish Interoperability Profile (.json) files.</td>
</tr>
<tr>
<td>redfish.dmtf.org/profiles/archive</td>
<td>Subfolders contain profile files specific to a particular profile version or release.</td>
</tr>
<tr>
<td>redfish.dmtf.org/dictionaries</td>
<td>Durable URL for programmatic access to all v1.xx Redfish Device Enablement Dictionary files.</td>
</tr>
<tr>
<td>redfish.dmtf.org/</td>
<td>Durable URL for programmatic access to all v1.xx Redfish Device Enablement Dictionary files.</td>
</tr>
</tbody>
</table>
11. Schema definition languages

Individual resources and their dependent types and actions are defined within a Redfish schema document. This clause describes how these documents are constructed in the following formats:

- OData Common Schema Definition Language
- JSON Schema
- OpenAPI

11.1. OData Common Schema Definition Language

OData Common Schema Definition Language (CSDL) is an XML schema format defined by the OData CSDL Specification. The following clause describes how Redfish uses CSDL to describe resources and resource collections.

11.1.1. File naming conventions for CSDL

Redfish CSDL schema files shall be named using the resource name value, followed by _v and the major version of the schema. Because a single CSDL schema file contains all minor revisions of the schema, only the major version appears in the file name. The file name shall be formatted as:

TypeName_v*MajorVersion*.xml

For example, version 1.3.0 of the Chassis schema is Chassis_v1.xml.

11.1.2. Core CSDL files

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RedfishError_v1.xml</td>
<td>Payload definition of the Redfish error response.</td>
</tr>
<tr>
<td>RedfishExtensions_v1.xml</td>
<td>All definitions for Redfish types and annotations.</td>
</tr>
<tr>
<td>Resource_v1.xml</td>
<td>All base definitions for resources, resource collections, and</td>
</tr>
</tbody>
</table>
common properties, such as Status.

11.1.3. CSDL format

The outer element of the OData schema representation document shall be the Edmx element, and shall have a Version attribute with a value of 4.0.

```
<edmx:Edmx
  xmlns:edmx="http://docs.oasis-open.org/odata/ns/edmx"
  Version="4.0">
 <!-- edmx:Reference and edmx:DataService elements go here -->
</edmx:Edmx>
```

The Referencing other CSDL files and CSDL data services clauses describe the items that are found within the Edmx element.

11.1.3.1. Referencing other CSDL files

CSDL files may reference types defined in other CSDL documents. This is done by including Reference tags.

The Reference element uses the Uri attribute to specify a CSDL file. The Reference element also contains one or more Include tags that specify the Namespace attribute containing the types to be referenced, along with an optional Alias attribute for that namespace.

Type definitions generally reference the OData and Redfish namespaces for common type annotation terms. Redfish CSDL files always use the Alias attribute on the following namespaces:

- Org.OData.Core.V1 is aliased as OData.
- Org.OData.Measures.V1 is aliased as Measures.
- RedfishExtensions.v1_0_0 is aliased as Redfish.
- Validation.v1_0_0 is aliased as Validation.

```
<edmx:Reference
  Uri="http://docs.oasis-open.org/odata/odata/v4.0/cs01/vocabularies/Org.OData.Core.V1.xml">
  <edmx:Include
    Namespace="Org.OData.Core.V1" Alias="OData"/>
</edmx:Reference>

<edmx:Reference
  <edmx:Include
</edmx:Reference>
```
11.1.3.2. CSDL data services

Define structures, enumerations, and other definitions in CSDL within a namespace. Use a Schema tag to define the schema and use the Namespace attribute to declare the name of the namespace.

Redfish uses namespaces to differentiate different versions of the schema. CSDL enables structures to inherit from other structures, which enables newer namespaces to define only the changes. The Elements of CSDL namespaces clause describes this behavior.

The Schema element is a child of the DataServices element, which is a child of the Edmx element:

```
<edmx:DataServices>
  <Schema xmlns="http://docs.oasis-open.org/odata/ns/edm" Namespace="MyTypes.v1_0_0">
    <!-- Type definitions for version 1.0.0 of MyTypes go here -->
  </Schema>
  <Schema xmlns="http://docs.oasis-open.org/odata/ns/edm" Namespace="MyTypes.v1_1_0">
    <!-- Type definitions for version 1.1.0 of MyTypes go here -->
  </Schema>
</edmx:DataServices>
```

11.1.4. Elements of CSDL namespaces

The following clauses describe the definitions within each namespace:

- Qualified names
- Entity type and complex type elements

11.1.4.1. Qualified names

Many definitions in CSDL use references to qualified names. CSDL defines this as a string in the form:

```
Namespace.TypeName
```

where
For example, if a reference is made to `MyType.v1_0_0.MyDefinition`, the definition can be found in the `MyType.v1_0_0` namespace with an element named `MyDefinition`.

### 11.1.4.2. Entity type and complex type elements

Use the `EntityType` and `ComplexType` tags to define the entity type and complex type elements, respectively. These elements define a JSON structure and their set of properties. This is done by defining `property elements` and `navigation property elements` within the `EntityType` or `ComplexType` tags.

All entity types and complex types contain a `Name` attribute, which specifies the name of the definition.

Entity types and complex types may have a `BaseType` attribute, which specifies a qualified name. When the `BaseType` attribute is used, all definitions of the referenced `BaseType` are available to the entity type or complex type being defined.

All resources and resource collections are defined with the entity type element. Resources inherit from `Resource.v1_0_0.Resource`, and resource collections inherit from `Resource.v1_0_0.ResourceCollection`.

Most structured properties are defined with the complex type element. Some use the entity type element that inherits from `Resource.v1_0_0.ReferenceableMember`. The entity type element enables references to be made by using the `Navigation Property element`, whereas the complex type element does not allow for this usage.

Example entity type and complex type element:

```xml
<EntityType Name="TypeA" BaseType="Resource.v1_0_0.Resource">
  <Annotation Term="OData.Description" String="The TypeA entity type description."/>
  <Annotation Term="OData.LongDescription" String="The TypeA entity type normative description."/>
  <!-- Property and navigation property definitions go here -->
</EntityType>

<ComplexType Name="PropertyTypeA">
  <Annotation Term="OData.Description" String="The TypeA structured property description."/>
  <Annotation Term="OData.LongDescription" String="The TypeA structured property normative description."/>
  <!-- Property and navigation property definitions go here -->
</ComplexType>
```
11.1.4.3. Action element

Use the Action tag to define the action element. This element defines an action that can be performed on a resource.

All action elements contain a Name attribute, which specifies the name of the action. The action shall be represented in payloads as the qualified name of the action, preceded by #.

In Redfish, all action elements contain the IsBound attribute that is always set to true, which indicates that the action appears as a member of a structured type.

The action element contains one or more Parameter tags that specify the Name and Type of each parameter.

Because all action elements in Redfish use the term IsBound="true", the first parameter is called the "binding parameter" and specifies the structured type to which the action belongs. In Redfish, this is always going to be one of the following complex type elements:

- For standard actions, the Actions complex type for the resource.
- For OEM actions, the OemActions complex type for the resource.

The remaining Parameter elements describe additional parameters to be passed to the action. Parameters containing the term Nullable="false" are required to be provided in the action request.

Some action parameters may specify a type that is defined by an entity type element. In these cases, the parameter in the request will be a reference object to a resource within the service.
11.1.4.3.1. Action element for OEM actions

OEM-specific actions shall be defined by using the action element with the binding parameter set to the OemActions complex type for the resource. For example, the following definition defines the OEM #Contoso.Ping action for a ComputerSystem.

```xml
<Schema xmlns="http://docs.oasis-open.org/odata/ns/edm" Namespace="Contoso">
  <Action Name="Ping" IsBound="true">
    <Parameter Name="ComputerSystem" Type="ComputerSystem.v1_0_0.OemActions"/>
  </Action>
</Schema>
```

11.1.4.3.2. Action with a response body

A response body for an action shall be defined using the ReturnType tag within an Action element. For example, the following definition defines the GenerateCSR action with a response that contains the definition specified by GenerateCSRResponse.

```xml
<Action Name="GenerateCSR" IsBound="true">
  <Parameter Name="CertificateService" Type="CertificateService.v1_0_0.Actions"/>
  ...
  <ReturnType Type="CertificateService.v1_0_0.GenerateCSRResponse" Nullable="false"/>
</Action>

<ComplexType Name="GenerateCSRResponse">
  <Annotation Term="OData.AdditionalProperties" Bool="false"/>
  <Annotation Term="OData.Description" String="The response body for the GenerateCSR action."/>
  <NavigationProperty Name="CertificateCollection" Type="CertificateCollection.CertificateCollection" Nullable="false">
    <Annotation Term="OData.Permissions" EnumMember="OData.Permission/Read"/>
    <Annotation Term="OData.Description" String="The link to the certificate resource collection where the certificate is installed."/>
    <Annotation Term="Redfish.Required"/>
  </NavigationProperty>
  <Property Name="CSRString" Type="Edm.String" Nullable="false">
    <Annotation Term="OData.Permissions" EnumMember="OData.Permission/Read"/>
    <Annotation Term="OData.Description" String="The string for the certificate signing request."/>
  </Property>
</ComplexType>
```

Using the above example, the following payload is an example response for the GenerateCSR action.
11.1.4.4. Property element

Properties of resources, resource collections, and structured properties are defined using the property element. The Property tag defines a property element inside entity type and complex type elements.

All property elements contain a Name attribute, which specifies the name of the property.

All property elements contain a Type attribute specifies the data type. The Type attribute shall be one of the following names or types:

- A qualified name that references an enum type element.
- A qualified name that references a complex type element.
- A primitive data type.
- An array of the previous names or types by using the Collection term.

Primitive data types shall be one of the following:

<table>
<thead>
<tr>
<th>Type</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>Edm.Boolean</td>
<td>True or False.</td>
</tr>
<tr>
<td>Edm.DateTimeOffset</td>
<td>Date-time string.</td>
</tr>
<tr>
<td>Edm.Decimal</td>
<td>Numeric values with fixed precision and scale.</td>
</tr>
<tr>
<td>Edm.Double</td>
<td>IEEE 754 binary64 floating-point number (15-17 decimal digits).</td>
</tr>
<tr>
<td>Edm.Duration</td>
<td>Duration string.</td>
</tr>
<tr>
<td>Edm.Guid</td>
<td>GUID/UUID string.</td>
</tr>
<tr>
<td>Edm.Int64</td>
<td>Signed 64-bit integer.</td>
</tr>
<tr>
<td>Edm.String</td>
<td>UTF-8 string.</td>
</tr>
</tbody>
</table>

Property elements may specify a Nullable attribute. If the attribute is false, the property value cannot be null. If the attribute is true or absent, the property value can be null.
Example property element:

```xml
<Property Name="Property1" Type="Edm.String" Nullable="false">
  <Annotation Term="OData.Description" String="The Property1 property description."
  />
  <Annotation Term="OData.LongDescription" String="The Property1 property normative
description." />
  <Annotation Term="OData.Permissions" EnumMember="OData.Permission/Read" />
  <Annotation Term="Redfish.Required" />
  <Annotation Term="Measures.Unit" String="Watts" />
</Property>
```

11.1.4.5. Navigation property element

- **Reference properties** of resources, resource collections, and structured properties are defined using the navigation property element. The `NavigationProperty` tag defines a navigation property element inside entity type and complex type elements.

All navigation property elements contain a `Name` attribute, which specifies the name of the property.

All navigation property elements contain a `Type` attribute specifies the data type. The `Type` attribute is a qualified name that references an entity type element. This can also be made into an array using the Collection term.

Navigation property elements may specify a `Nullable` attribute. If the attribute is `false`, the property value cannot be null. If the attribute is `true` or absent, the property value can be null.

Unless the reference property is to be expanded, all navigation properties in Redfish use the `OData.AutoExpandReferences` annotation element to show that the reference is always available.

Example navigation property element:

```xml
<NavigationProperty Name="RelatedType" Type="MyTypes.TypeB">
  <Annotation Term="OData.Description" String="The RelatedType navigation property
description." />
  <Annotation Term="OData.LongDescription" String="The RelatedType navigation
property normative description." />
  <Annotation Term="OData.AutoExpandReferences" />
</NavigationProperty>
```

11.1.4.6. Enum type element

- Use the `EnumType` tag to define the enum type element. This element defines a set of enumeration values, which may be applied to one or more properties.
All enum type elements contain a Name attribute, which specifies the name of the set of enumeration values.

Enum type elements contain Member tags that define the members of the enumeration. The Member tags contain a Name attribute that specifies the string value of the member name.

```xml
<EnumType Name="EnumTypeA">
  <Annotation Term="OData.Description" String="The EnumTypeA enum type description.">
    <Annotation Term="OData.LongDescription" String="The EnumTypeA enum type normative description."/>
    <Member Name="MemberA">
      <Annotation Term="OData.Description" String="The description of MemberA"/>
    </Member>
    <Member Name="MemberB">
      <Annotation Term="OData.Description" String="The description of MemberB"/>
    </Member>
  </Annotation>
</EnumType>
```

### 11.1.4.7. Annotation element

Annotations in CSDL are expressed using the Annotation element. The Annotation element can be applied to any schema element in CSDL.

The following examples show how each Redfish schema annotation is expressed in CSDL.

- The OData Core Schema defines terms with the OData prefix.
- The OData Measures Schema defines terms with the Measures prefix.
- The RedfishExtensions Schema defines terms with the Redfish prefix.

Example description annotation:

```xml
<Annotation Term="OData.Description" String="This property contains the user name for the account."/>
```

Example long description annotation:

```xml
<Annotation Term="OData.LongDescription" String="This property shall contain the user name for the account."/>
```

Example additional properties annotation:
Example **permissions annotation** (read-only):

```xml
<Annotation Term="OData.AdditionalProperties"/>
```

Example **permissions annotation** (read/write):

```xml
<Annotation Term="OData.AdditionalProperties" EnumMember="OData.Permission/Read"/>
```

Example **permissions annotation** (read/write):

```xml
<Annotation Term="OData.AdditionalProperties" EnumMember="OData.Permission/ReadWrite"/>
```

Example **required annotation**:

```xml
<Annotation Term="Redfish.Required"/>
```

Example **required on create annotation**:

```xml
<Annotation Term="Redfish.RequiredOnCreate"/>
```

Example **units of measure annotation**:

```xml
<Annotation Term="Measures.Unit" String="MiBy"/>
```

Example **expanded resource annotation**:

```xml
<Annotation Term="OData.AutoExpand"/>
```

Example **insert capabilities annotation** (showing POST is not allowed):

```xml
<Annotation Term="Capabilities.InsertRestrictions">
  <Record>
    <PropertyValue Property="Insertable" Bool="false"/>
  </Record>
</Annotation>
```
Example **update capabilities annotation** (showing PATCH and PUT are allowed):

```
<Annotation Term="Capabilities.UpdateRestrictions">
  <Record>
    <PropertyValue Property="Updatable" Bool="true"/>
    <Annotation Term="OData.Description" String="Manager accounts can be updated to change the password and other writable properties."/>
  </Record>
</Annotation>
```

Example **delete capabilities annotation** (showing DELETE is allowed):

```
<Annotation Term="Capabilities.DeleteRestrictions">
  <Record>
    <PropertyValue Property="Deletable" Bool="true"/>
    <Annotation Term="OData.Description" String="Manager accounts are removed with a Delete operation."/>
  </Record>
</Annotation>
```

Example **resource URI patterns annotation**:

```
<Annotation Term="Redfish.Uris">
  <Collection>
    <String>/redfish/v1/AccountService/Accounts/{ManagerAccountId}</String>
  </Collection>
</Annotation>
```

Example **owning entity annotation**:

```
<Annotation Term="Redfish.OwningEntity" String="DMTF"/>
```

Example **deprecated annotation**:

```
<Annotation Term="Redfish.Revisions">
  <Collection>
    <Record>
      <PropertyValue Property="Kind" EnumMember="Redfish.RevisionKind/Deprecated"/>
      <PropertyValue Property="Version" String="v1_3_0"/>
      <PropertyValue Property="Description" String="This property has been deprecated."/>
    </Record>
  </Collection>
</Annotation>
```
11.2. JSON Schema

The [JSON Schema Specification](https://json-schema.org) defines a JSON format for describing JSON payloads. The following clause describes how Redfish uses JSON Schema to describe resources and resource collections.

11.2.1. File naming conventions for JSON Schema

Versioned Redfish JSON Schema files shall use the resource name to name the file, in this format:

```json
ResourceName.vMajorVersion_MinorVersion_Errata.json
```

For example, version 1.3.0 of the Chassis schema is `Chassis.v1_3_0.json`.

Unversioned Redfish JSON Schema files shall use the resource name to name the file, in this format:

```json
ResourceName.json
```

For example, the unversioned definition of the Chassis schema is `Chassis.json`.

11.2.2. Core JSON Schema files

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>odata-v4.json</td>
<td>Definitions for common OData properties.</td>
</tr>
<tr>
<td>redfish-error.v1_0_0.json and its subsequent versions</td>
<td>Payload definition of the <a href="https://docs.oasis-open.org/odata/odata/v4.0/odata-v4-fullspec.html">Redfish error response</a>.</td>
</tr>
<tr>
<td>redfish-schema-v1.json</td>
<td>Extensions to the JSON Schema that define Redfish JSON Schema files.</td>
</tr>
<tr>
<td>Resource.json and its subsequent versions</td>
<td>All base definitions for resources, resource collections, and common properties, such as Status.</td>
</tr>
</tbody>
</table>

11.2.3. JSON Schema format

Each JSON Schema file contains a JSON object to describe resources, resource collections, and other definitions for the data model.
The JSON object contains the following terms:

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$id</td>
<td>Reference to the URI where the schema file is published.</td>
</tr>
<tr>
<td>$ref</td>
<td>For a schema file that describes a resource or resource collection, the reference to the structural definition of the resource or resource collection.</td>
</tr>
<tr>
<td>copyright</td>
<td>Copyright statement for the organization producing the JSON Schema.</td>
</tr>
<tr>
<td>definitions</td>
<td>Structures, enumerations, and other definitions defined by the schema.</td>
</tr>
<tr>
<td>title</td>
<td>For a schema file that describes a resource or resource collection, the matching type identifier for the resource or resource collection.</td>
</tr>
</tbody>
</table>

11.2.4. JSON Schema definitions body

This clause describes the types of definitions found in the definitions term of a Redfish JSON Schema file.

11.2.4.1. Resource definitions in JSON Schema

To satisfy versioning requirements, the JSON Schema representation of each resource has one unversioned schema file, and a set of versioned schema files.

The unversioned definition of a resource contains an anyOf statement. This statement consists of an array of $ref terms, which point to the following definitions:

- The JSON Schema definition for a reference property.
- The versioned definitions of the resource.

The unversioned definition of a resource also uses the uris term to express the allowable URIs for the resource, and the deletable, insertable, and updatable terms to express the capabilities of the resource.

The following example shows an unversioned resource definition in JSON Schema:

```json
{
  "ComputerSystem": {
    "anyOf": [
```
The versioned definition of a resource contains the property definitions for the given version of the resource.

### 11.2.4.2. Enumerations in JSON Schema

Definitions for enumerations can consist of these terms:

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>enum</td>
<td>String array that contains the possible enumeration values.</td>
</tr>
<tr>
<td>enumDescriptions</td>
<td>Object that contains the descriptions for each of the enumerations as name-value pairs.</td>
</tr>
<tr>
<td>enumLongDescriptions</td>
<td>Object that contains the long descriptions for each of the enumerations as name-value pairs.</td>
</tr>
<tr>
<td>Term</td>
<td>Description</td>
</tr>
<tr>
<td>----------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>enumerations as name-value pairs.</td>
<td></td>
</tr>
<tr>
<td>enumDeprecated</td>
<td>Object that contains the deprecation guidance for each of the enumerations as name-value pairs.</td>
</tr>
<tr>
<td>enumVersionDeprecated</td>
<td>Object that contains the deprecation version information for each of the enumerations as name-value pairs.</td>
</tr>
<tr>
<td>type</td>
<td>Because all enumerations in Redfish are strings, the type term always has the string value.</td>
</tr>
</tbody>
</table>

The following example shows an enumeration definition in JSON Schema:

```json
{
   "IndicatorLED": {
      "enum": [
         "Lit",
         "Blinking",
         "Off"
      ],
      "enumDescriptions": {
         "Blinking": "The Indicator LED is blinking.",
         "Lit": "The Indicator LED is lit.",
         "Off": "The Indicator LED is off."
      },
      "enumLongDescriptions": {
         "Blinking": "This value shall represent the Indicator LED is in a blinking state where the LED is being turned on and off in repetition.",
         "Lit": "This value shall represent the Indicator LED is in a solid on state.",
         "Off": "This value shall represent the Indicator LED is in a solid off state."
      },
      "type": "string"
   },
   ...
}
```

### 11.2.4.3. Actions in JSON Schema

Versioned definitions of resources contain a definition called Actions. This definition is a container with a set of terms that point to the different actions supported by the resource. The names of standard actions shall be in the form:
Another definition within the same schema file describes the action itself. This definition contains a term called parameters to describe the client request body. It also contains property definitions for the target and title properties shown in response payloads for the resource.

The following example shows a definition of an action:

```json
{
    "Reset": {
        "additionalProperties": false,
        "description": "This action resets the system.",
        "longDescription": "This action shall perform a reset of the ComputerSystem.",
        "parameters": {
            "ResetType": {
                "$ref": "http://redfish.dmtf.org/schemas/v1/Resource.json#/definitions/
ResetType",
                "description": "The type of reset to be performed.",
                "longDescription": "This parameter shall define the type of reset to
be performed."
            }
        },
        "properties": {
            "target": {
                "description": "Link to invoke action",
                "format": "uri",
                "type": "string"
            }
        }
    }
}
```
Some action parameters may specify a type that is a resource definition. In these cases, the parameter in the request will be a reference object to a resource within the service.

**11.2.4.3.1. OEM actions in JSON Schema**

OEM-specific actions shall be defined by using an action definition in an appropriately named JSON Schema file. For example, the following definition defines the OEM #Contoso.Ping action, assuming it's found in the versioned Contoso JSON Schema file, such as Contoso.v1_0_0.json.

```
{
   "Ping": {
      "additionalProperties": false,
      "parameters": {},
      "properties": {
         "target": {
            "description": "Link to invoke action",
            "type": "string"
         },
         "title": {
            "description": "Friendly action name",
            "type": "string"
         }
      },
      "type": "object"
   },
   ...}
```

**11.2.4.3.2. Action with a response body**

A response body for an action shall be defined using the actionResponse term within the action definition. For example, the following definition defines the GenerateCSR action with a response that
contains the definition specified by `#/definitions/GenerateCSRResponse`.

```json
{
  "GenerateCSR": {
    "actionResponse": {
      "$ref": "#/definitions/GenerateCSRResponse",
    },
    "parameters": {}
  },
  "GenerateCSRResponse": {
    "additionalProperties": false,
    "description": "The response body for the GenerateCSR action.",
    "properties": {
      "CSRString": {
        "description": "The string for the certificate signing request.",
        "readonly": true,
        "type": "string"
      },
      "CertificateCollection": {
        "$ref": "http://redfish.dmtf.org/schemas/v1/CertificateCollection.json#/definitions/CertificateCollection",
        "description": "The link to the certificate resource collection where the certificate is installed.",
        "readonly": true
      }
    },
    "required": [
      "CertificateCollection",
      "CSRString"
    ],
    "type": "object"
  }
}
```

In the previous example, the following payload is an example response for the GenerateCSR action.

```json
{
  "CSRString": "-----BEGIN CERTIFICATE REQUEST-----...
  "CertificateCollection": {
    "$odata.id": "/redfish/v1/Managers/BMC/NetworkProtocol/HTTPS/Certificates"
  }
}
```
11.2.5. JSON Schema terms

Redfish uses the following JSON Schema terms to provide [schema annotations](#) for Redfish JSON Schema:

<table>
<thead>
<tr>
<th>JSON Schema term</th>
<th>Related Redfish schema annotation</th>
</tr>
</thead>
<tbody>
<tr>
<td>description</td>
<td>Description</td>
</tr>
<tr>
<td>enumDescriptions</td>
<td></td>
</tr>
<tr>
<td>longDescription</td>
<td>Long description</td>
</tr>
<tr>
<td>enumLongDescriptions</td>
<td></td>
</tr>
<tr>
<td>additionalProperties</td>
<td>Additional properties</td>
</tr>
<tr>
<td>readonly</td>
<td>Permissions</td>
</tr>
<tr>
<td>required</td>
<td>Required</td>
</tr>
<tr>
<td>requiredOnCreate</td>
<td>Required on create</td>
</tr>
<tr>
<td>units</td>
<td>Units of measure</td>
</tr>
<tr>
<td>autoExpand</td>
<td>Expanded resource</td>
</tr>
<tr>
<td>deletable</td>
<td>Resource capabilities</td>
</tr>
<tr>
<td>insertable</td>
<td></td>
</tr>
<tr>
<td>updatable</td>
<td></td>
</tr>
<tr>
<td>uris</td>
<td>Resource URI patterns</td>
</tr>
<tr>
<td>owningEntity</td>
<td>Owning entity</td>
</tr>
<tr>
<td>deprecated</td>
<td>Deprecated</td>
</tr>
<tr>
<td>versionDeprecated</td>
<td></td>
</tr>
</tbody>
</table>

11.3. OpenAPI

The [OpenAPI Specification](https://spec.openapis.org/oas/v3.0.2) defines a format for describing JSON payloads and the set of URIs a client can access on a service. The following clause describes how Redfish uses OpenAPI to describe resources and resource collections.

11.3.1. File naming conventions for OpenAPI schema

Versioned Redfish OpenAPI files shall be named using the [resource name](#), following the format:
For example, version 1.3.0 of the Chassis schema is Chassis.v1_3_0.yaml.

Unversioned Redfish OpenAPI files shall use the resource name to name the file, in this format:

ResourceName.yaml

For example, the unversioned definition of the Chassis schema is Chassis.yaml.

### 11.3.2. Core OpenAPI schema files

<table>
<thead>
<tr>
<th>File</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>odata-v4.yaml</td>
<td>Definitions for common OData properties.</td>
</tr>
<tr>
<td>openapi.yaml</td>
<td>URI paths and their respective payload structures.</td>
</tr>
<tr>
<td>Resource.yaml and its</td>
<td>All base definitions for resources, resource collections, and</td>
</tr>
<tr>
<td>subsequent versions</td>
<td>common properties, such as Status.</td>
</tr>
</tbody>
</table>

### 11.3.3. openapi.yaml

The openapi.yaml file is the starting point for clients to understand the construct of the service. It contains the following terms:

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>components</td>
<td>Global definitions. For Redfish, contains the format of the Redfish error response.</td>
</tr>
<tr>
<td>info</td>
<td>Structure consisting of information about what the openapi.yaml is describing, such as the author of the file and any contact information.</td>
</tr>
<tr>
<td>openapi</td>
<td>Version of OpenAPI the document follows.</td>
</tr>
<tr>
<td>paths</td>
<td>URIs supported by the document, with possible methods, response bodies, and request bodies.</td>
</tr>
</tbody>
</table>

The service shall return the openapi.yaml, if present in the Redfish Service, as a YAML document by using either the application/yaml or application/vnd.oai.openapi MIME types. The service may append ;charset=utf-8 to the MIME type. Note that while the application/yaml type is in common use today, the application/vnd.oai.openapi type was recently defined and approved specifically to support OpenAPI. Implementations should use caution when selecting the MIME type as this specification may change in the future to reflect adoption of the OpenAPI-specific MIME type.
The `paths` term contains an array of the possible URIs. For each URI, it also lists the possible methods. For each method, it lists the possible response bodies and request bodies.

Example `paths` entry for a resource:

```
/redfish/v1/Systems/{ComputerSystemId}:
  get:
    parameters:
      - description: The value of the Id property of the ComputerSystem resource
        in: path
        name: ComputerSystemId
        required: true
        schema:
          type: string
    responses:
      '200':
        content:
          application/json:
            schema:
              $ref: http://redfish.dmtf.org/schemas/v1/ComputerSystem.v1_6_0.yaml#/components/schemas/ComputerSystem
        description: The response contains a representation of the ComputerSystem resource
      default:
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/RedfishError'
        description: Error condition
```

Example `paths` entry for an action:

```
/redfish/v1/Systems/{ComputerSystemId}/Actions/ComputerSystem.Reset:
  post:
    parameters:
      - description: The value of the Id property of the ComputerSystem resource
        in: path
        name: ComputerSystemId
        required: true
        type: string
    requestBody:
      content:
        application/json:
          schema:
            $ref: http://redfish.dmtf.org/schemas/v1/
```
11.3.4. OpenAPI file format

With the exception of `openapi.yaml`, each OpenAPI file contains a YAML object to describe resources, resource collections, or other definitions for the data model. The YAML object contains the following terms:

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>components</td>
<td>Structures, enumerations, and other definitions defined by the schema.</td>
</tr>
<tr>
<td>x-copyright</td>
<td>Copyright statement for the organization producing the OpenAPI file.</td>
</tr>
<tr>
<td>title</td>
<td>For a schema file that describes a resource or resource collection, the matching type identifier for the resource or resource collection.</td>
</tr>
</tbody>
</table>

11.3.5. OpenAPI components body

This clause describes the types of definitions that can be found in the `components` term of a Redfish
OpenAPI file.

11.3.5.1. Resource definitions in OpenAPI

To satisfy versioning requirements, the OpenAPI representation of each resource has one unversioned schema file, and a set of versioned schema files.

The unversioned definition of a resource contains an anyOf statement. This statement consists of an array of $ref terms, which point to the following definitions:

- The OpenAPI definition for a reference property.
- The versioned definitions of the resource.

Example unversioned resource definition in OpenAPI:

```yaml
ComputerSystem:
  anyOf:
  - $ref: http://redfish.dmtf.org/schemas/v1/odata.v4_0_3.yaml#/components/schemas/idRef
  - $ref: http://redfish.dmtf.org/schemas/v1/ComputerSystem.v1_0_0.yaml#/components/schemas/ComputerSystem
  - $ref: http://redfish.dmtf.org/schemas/v1/ComputerSystem.v1_0_1.yaml#/components/schemas/ComputerSystem
  - $ref: http://redfish.dmtf.org/schemas/v1/ComputerSystem.v1_6_0.yaml#/components/schemas/ComputerSystem
  description: The ComputerSystem schema represents a general purpose machine or system.
  x-longDescription: This resource shall be used to represent resources that represent a computing system.
```

The versioned definition of a resource contains the property definitions for the given version of the resource.

11.3.5.2. Enumerations in OpenAPI

Definitions for enumerations can consist of the following terms:

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>enum</td>
<td>String array that contains the possible enumeration values.</td>
</tr>
<tr>
<td>type</td>
<td>Because all enumerations in Redfish are strings, the type term always has the value string.</td>
</tr>
<tr>
<td>x-enumDescriptions</td>
<td>Object that contains the descriptions for each of the enumerations</td>
</tr>
<tr>
<td>Term</td>
<td>Description</td>
</tr>
<tr>
<td>---------------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td>as name-value pairs.</td>
</tr>
<tr>
<td>x-enumLongDescriptions</td>
<td>Object that contains the long descriptions for each enumeration as a name-value pair.</td>
</tr>
<tr>
<td>x-enumDeprecated</td>
<td>Object that contains the deprecation guidance for each of the enumerations as name-value pairs.</td>
</tr>
<tr>
<td>x-enumVersionDeprecated</td>
<td>Object that contains the deprecation version information for each of the enumerations as name-value pairs.</td>
</tr>
</tbody>
</table>

Example enumeration definition in OpenAPI:

```json
IndicatorLED:
  enum:
    - Lit
    - Blinking
    - 'Off'
  type: string
  x-enumDescriptions:
    Blinking: The Indicator LED is blinking.
    Lit: The Indicator LED is lit.
    'Off': The Indicator LED is off.
  x-enumLongDescriptions:
    Blinking: This value shall represent the Indicator LED is in a blinking state where the LED is being turned on and off in repetition.
    Lit: This value shall represent the Indicator LED is in a solid on state.
    'Off': This value shall represent the Indicator LED is in a solid off state.
```

11.3.5.3. Actions in OpenAPI

Versioned definitions of resources contain a definition called Actions. This definition is a container with a set of terms that point to the different actions supported by the resource. The names of standard actions shall be in the form:

```
#ResourceType.ActionName
```

Example Actions definition:

```json
Actions:
  additionalProperties: false
```
description: The available actions for this resource.
properties:
  '#ComputerSystem.Reset':
    $ref: '#/components/schemas/Reset'
type: object
x-longDescription: This type shall contain the available actions for this resource.

Another definition within the same schema file describes the action itself. This definition contains property definitions for the target and title properties shown in response payloads for the resource.

The following example shows a definition of an action:

Reset:
  additionalProperties: false
description: This action resets the system.
properties:
  target:
    description: Link to invoke action
    format: uri
    type: string
  title:
    description: Friendly action name
    type: string
type: object
x-longDescription: This action shall reset the ComputerSystem.

The parameters for the action are shown in another definition with RequestBody appended to the name of the action. This gets mapped from the openapi.yaml file for expressing the POST method for the URI of the action.

The following example shows a definition of parameters of an action:

ResetRequestBody:
  additionalProperties: false
description: This action resets the system.
properties:
  ResetType:
    $ref: http://redfish.dmtf.org/schemas/v1/Resource.yaml#/components/schemas/
    ResetType
description: The reset type.
x-longDescription: This parameter shall define the type of reset to perform.
type: object
x-longDescription: This action shall reset the ComputerSystem.
11.3.5.3.1. OEM actions in OpenAPI

OEM-specific actions shall be defined by using an action definition in an appropriately named OpenAPI file. For example, the following definition defines the OEM #Contoso.Ping action, assuming it’s found in the versioned Contoso OpenAPI file with a name, such as Contoso.v1_0_0.yaml.

```
Ping:
  additionalProperties: false
  properties:
    target:
      description: Link to invoke action
      format: uri
      type: string
    title:
      description: Friendly action name
      type: string
  type: object

PingRequestBody:
  additionalProperties: false
  properties: {}
  type: object
```

11.3.6. OpenAPI terms used by Redfish

The following OpenAPI terms provide schema annotations for Redfish OpenAPI files:

<table>
<thead>
<tr>
<th>OpenAPI term</th>
<th>Related Redfish schema annotation</th>
</tr>
</thead>
<tbody>
<tr>
<td>description</td>
<td>Description</td>
</tr>
<tr>
<td>x-enumDescriptions</td>
<td></td>
</tr>
<tr>
<td>x-longDescription</td>
<td>Long description</td>
</tr>
<tr>
<td>x-enumLongDescriptions</td>
<td></td>
</tr>
<tr>
<td>additionalProperties</td>
<td>Additional properties</td>
</tr>
<tr>
<td>readOnly</td>
<td>Permissions</td>
</tr>
<tr>
<td>required</td>
<td>Required</td>
</tr>
<tr>
<td>x-requiredOnCreate</td>
<td>Required on create</td>
</tr>
<tr>
<td>x-units</td>
<td>Units of measure</td>
</tr>
<tr>
<td>x-autoExpand</td>
<td>Expanded resource</td>
</tr>
</tbody>
</table>
11.4. Schema modification rules

Schema referenced from the implementation may vary from the canonical definitions of those schema defined by the Redfish schema or other entities, provided they adhere to the following rules. Clients should take this into consideration when attempting operations on the resources defined by schema.

- Modified schema may constrain a read/write property to be read only.
- Modified schema may constrain the capabilities of a resource or resource collection to remove support for HTTP operations.
- Modified schema may remove properties.
- Modified schema may change any external references to point to Redfish schema that adheres to the modification rules.
- Modified schema may change the owning entity annotation to specify who made the modifications.
- Modified schema may add URIs to the resource URI patterns annotation to define OEM URIs for standard resources and shall follow the OEM URI rules specified by the OEM resource naming and URIs clause.
- Other modifications to the schema shall not be allowed.

12. Service details

12.1. Eventing

This clause describes how to use the REST-based mechanism to subscribe to and receive event messages.

**Note:** For security implications of eventing, see the Security details clause.

The Redfish Service requires a client or administrator to create subscriptions to receive events.

To create a subscription, use one of these methods:

- Directly HTTP POST to the subscription collection.
- Indirectly open a Server-Sent Events (SSE) connection for the Event Service.
12.1.1. POST to subscription collection

To locate the Event Service, the client traverses the Redfish Service interface. The Event Service is located in the Service Root, as described in the ServiceRoot schema.

Once the client discovers the service, they send an HTTP POST to the resource collection URI for Subscriptions in the Event Service to subscribe to events. For the subscription body syntax, see the Redfish EventDestination schema. This request includes:

- The URI where an event-receiver client expects events to be sent. When an event is triggered within the Redfish Service, the service sends an event to that URI.
- The type of events to send.

If the subscription request succeeds, the service shall return:

- An HTTP 201 Created status code.
- The Location header that contains a URI of the newly created subscription resource.

If the subscription request succeeds, the service should return:

- A response body containing a representation of the subscription resource that conforms to the EventDestination schema.

After a subscription is registered with the service, clients begin receiving events. Clients do not receive events retroactively. The service does not retain historical events.

Services shall:

- Support push style eventing for all resources that can send events.
- Respond to a request to create a subscription with an error if the body of the request is conflicting. For instance, if parameters in the request are not supported, the service shall return the HTTP 400 Bad Request status code.
- Respond to a request to create a subscription with an error if the body of the request contains both RegistryPrefixes and MessageIds, and shall return the HTTP 400 Bad Request status code. These properties are considered mutually exclusive.
- Retain subscriptions as persistent across service restarts.

Services shall not:

- Push events by using HTTP POST unless an event subscription has been created. To terminate the event stream at any time, either the client or the service can delete the subscription.
- Send a push event payload larger than 1 Mebibyte (1 MiB). If there is more than 1 MiB worth of data to send the service shall divide the payload on the nearest Event entry such that the total payload transmitted to the client is less than 1 MiB. This restriction shall not apply to metric reports.
Services may:

- Terminate a subscription by sending a SubscriptionTerminated message from the Base Message Registry as the last event.
- Terminate a subscription if the number of delivery errors exceeds preconfigured thresholds.

To unsubscribe from the events associated with this subscription, the client or administrator shall send an HTTP DELETE request to the subscription’s resource URI.

Subsequent requests to subscription resources that have been terminated respond with the HTTP 404 Not Found status code.

Some configurable properties define the behavior for all event subscriptions. For details, see the Redfish EventService schema.

12.1.2. Open an SSE connection

A service may support the ServerSentEventUri property in the EventService resource. If a client performs a GET on the URI that the ServerSentEventUri contains, an SSE connection opens for the client. For details about this method, see the Server-Sent Events Event Service clause.

12.1.3. EventType-based eventing

DEPRECATED: EventType-based eventing is deprecated in the Redfish schema in favor of using RegistryPrefix and ResourceType.

DEPRECATED

A Redfish Service generates these types of events:

<table>
<thead>
<tr>
<th>Event</th>
<th>Occurs when</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Life cycle</td>
<td>Resources are created, modified, or destroyed. Usually indicates that the resource and, optionally, its properties have changed.</td>
<td>Not every modification of a resource results in an event. This behavior is similar to when ETags are changed and implementations may not send an event for every resource change. For example, if an event is sent for every Ethernet packet that is received or each time that a sensor changes one degree, more events than fit in a scalable interface are generated.</td>
</tr>
<tr>
<td>Event</td>
<td>Occurs when</td>
<td>Description</td>
</tr>
<tr>
<td>-----------</td>
<td>-----------------------------------------------------------------------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Alert</td>
<td>An event of some significance happens.</td>
<td>Usually adopts a Message Registry approach similar to extended error handling in that a <em>MessageId</em> is included.</td>
</tr>
<tr>
<td></td>
<td>Depending on the resource, may be generated directly or indirectly.</td>
<td>An example of an alert event is, a chassis is opened, a button is pushed, a cable is unplugged, or a threshold exceeded.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>These events usually do not correspond well to life cycle-type events. Therefore, alerts have their own category.</td>
</tr>
<tr>
<td>Metric report</td>
<td>The Telemetry Service generates or updates a metric report.</td>
<td>Generated as specified by the <em>MetricReportDefinition</em> resources found subordinate to the Telemetry Service. Can occur periodically, on demand, or when changes are detected in the metric properties.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>For details, see the Redfish <em>MetricReportDefinition</em> schema.</td>
</tr>
</tbody>
</table>

**END DEPRECATED**

**12.1.4. Subscribing to events**

To subscribe to events and filter received messages, a subscriber provides these properties:

<table>
<thead>
<tr>
<th>Property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RegistryPrefixes</td>
<td>An array of standard or OEM Message Registries.</td>
</tr>
<tr>
<td></td>
<td>An event is sent to the subscriber if one of the Message Registries that <em>RegistryPrefixes</em> lists defines the event message.</td>
</tr>
<tr>
<td></td>
<td>To receive messages from all registries, pass an empty array.</td>
</tr>
<tr>
<td></td>
<td>The contents of the array does not include the registry version.</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Property</td>
<td>Description</td>
</tr>
<tr>
<td>ResourceTypes</td>
<td>An array of standard or OEM resource types.</td>
</tr>
<tr>
<td></td>
<td>An event is sent to the subscriber if the <code>OriginOfCondition</code> resource type matches one of the <code>ResourceTypes</code> values.</td>
</tr>
<tr>
<td></td>
<td>The contents of the array does not include the schema version. For example, if the resource type is <code>Task.v1_2_0.Task</code>, the property value</td>
</tr>
<tr>
<td></td>
<td>is <code>Task</code>.</td>
</tr>
<tr>
<td></td>
<td>To receive messages from any resource, pass an empty array.</td>
</tr>
<tr>
<td>OriginResources</td>
<td>An array of URIs to resources.</td>
</tr>
<tr>
<td></td>
<td>An event is sent to the subscriber if the <code>OriginOfCondition</code> property matches one of the URIs listed in <code>OriginResources</code>.</td>
</tr>
<tr>
<td></td>
<td>To receive messages from any resource, pass an empty array.</td>
</tr>
<tr>
<td></td>
<td>To include subordinate resources regardless of depth, set the <code>SubordinateResources</code> property to <code>true</code>.</td>
</tr>
<tr>
<td>EventFormatType</td>
<td>The format that can be sent by using the <code>EventFormatTypes</code> property in the Event Service.</td>
</tr>
<tr>
<td></td>
<td>Represents the format of the payload sent to the event destination.</td>
</tr>
<tr>
<td></td>
<td>If the subscriber omits this value, the payload corresponds to the <code>Event</code> schema.</td>
</tr>
</tbody>
</table>

### 12.1.5. Event formats

The event formats are:
<table>
<thead>
<tr>
<th>Event format</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Metric report message</td>
<td>Used when the Telemetry Service generates a new or updates an existing metric report. Metric report message objects sent to the specified client endpoint shall contain the properties, as described in the Redfish MetricReport schema.</td>
</tr>
<tr>
<td>Event message objects</td>
<td>Used for all other types of events. Event message objects POSTed to the specified client endpoint shall contain the properties as described in the Redfish Event schema. Supports a Message Registry. In a Message Registry approach, a Message Registry lists the MessageIds in a well-known format. These MessageIds are terse in nature and thus they are much smaller than actual messages, making them suitable for embedded environments. The registry also contains a message. The message itself can have arguments and default values for severity and recommended actions. The MessageId property is in the format: RegistryName.MajorVersion.MinorVersion.MessageKey where</td>
</tr>
<tr>
<td></td>
<td>Event messages may also have an EventGroupId property, which lets clients know that different messages may be from the same event. For instance, if a LAN cable is disconnected, they</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>RegistryName</td>
<td>Pascal-cased name of the registry.</td>
</tr>
<tr>
<td>MajorVersion</td>
<td>Positive integer, which is the major version of the registry.</td>
</tr>
<tr>
<td>MinorVersion</td>
<td>Positive integer, which is the minor version of the registry.</td>
</tr>
<tr>
<td>MessageKey</td>
<td>Human-readable Pascal-cased key into the registry. Shall not include spaces, periods, or special characters.</td>
</tr>
</tbody>
</table>
12.1.6. OEM extensions

OEMs can extend both messages and Message Registries. Any individual message, per the `MessageRegistry` schema definition, define OEM sections. Thus, if OEMs wish to provide additional information or properties, use the OEM section.

OEMs shall not supply additional message arguments beyond those in a standard Message Registry. OEMs may substitute their own Message Registry for the standard registry to provide the OEM section within the registry but shall not change the standard values, such as messages, in such registries.

12.2. Asynchronous operations

Services that support asynchronous operations implement the `TaskService` and `Task` resources.

The Task Service describes the service that handles `task`. It contains a resource collection of zero or more `Task` resources. The `Task` resource describes a long-running operation that is spawned when a request takes longer than a few seconds, such as when a service is instantiated.

The `Task` schema defines task structure, including the start time, end time, task state, task status, and zero or more task-associated messages.

Each task has a number of possible states. The `Task` schema defines the exact states and their semantics.

When a client issues a request for a long-running operation, the service returns the HTTP `202 Accepted` status code and a `Location` header that contains the URI of the `task monitor` and, optionally, the `Retry-After` header that defines the amount of time that the client should wait before querying the status of the operation.

The task monitor is an opaque service-generated URI that the client who initiates the request can use. To query the status of an operation, and determine when the operation has been completed and whether it succeeded, the client performs a GET request on the task monitor. The client should not include the...
application/http MIME type in the Accept header.

The 202 Accepted response body should contain an instance of the Task resource that describes the state of the task.

As long as the operation is in process, the service shall continue to return the HTTP 202 Accepted status code when the client queries the task monitor URI.

If a service supports cancellation of a task, it shall have DELETE in the Allow header for the task monitor. To cancel the operation, the client may perform a DELETE on the task monitor URI. The service determines when to delete the associated Task resource.

To cancel the operation, the client may also perform a DELETE on the Task resource. Deleting the Task resource may invalidate the associated task monitor. A subsequent GET request on the task monitor URI returns either the HTTP 410 Gone status code or the HTTP 404 Not Found status code.

In the unlikely event that a DELETE of the task monitor or Task resource returns the HTTP 202 Accepted status code, an additional task shall not be started and instead the client may monitor the existing Task resource for the status of the cancellation request. When the task finally completes cancellation, operations to the task monitor and Task resources shall return the HTTP 404 Not Found status code.

After the operation has been completed, the service shall update the TaskState with the appropriate value. The Task schema defines the task completed values.

After the operation has been completed, the task monitor shall return:

- The appropriate HTTP status code, such as but not limited to 200 OK for most operations or 201 Created for POST to create a resource.
- The headers and response body of the initial operation, as if it had completed synchronously.

If the initial operation fails, the response body shall contain an error response.

If the operation has been completed and the service has already deleted the task, the service may return the HTTP 410 Gone or 404 Not Found status code. This situation can occur if the client waits too long to read the task monitor.

To continue to get status information, the client can use the resource identifier from the 202 Accepted response to directly query the Task resource.

- Services that support asynchronous operations shall implement the Task resource.
- The response to an asynchronous operation shall return the HTTP 202 Accepted status code and set the HTTP Location response header to the URI of a task monitor associated with the task. The response may also include the Retry-After header that defines the amount of time
that the client should wait before polling for status. The response body should contain a representation of the Task resource.

- GET requests to either the task monitor or Task resource shall return the current status of the operation without blocking.
- HTTP GET, PUT, and PATCH operations should always be synchronous.
- Clients shall be prepared to handle both synchronous and asynchronous responses for HTTP GET, PUT, PATCH, POST, and DELETE requests.
- Services shall persist pending tasks produced by client requests containing @Redfish.OperationApplyTime across service restarts, until the task begins execution.
- Tasks that are pending execution should include the @Redfish.OperationApplyTime property to indicate when the task will start. If the @Redfish.OperationApplyTime value is AtMaintenanceWindowStart or InMaintenanceWindowOnReset, the task should also include the @Redfish.MaintenanceWindow property.

12.3. Resource tree stability

The resource tree, which is defined as the set of URIs and array elements within the implementation, should be consistent on a single service across device resets or power cycles, and should withstand a reasonable amount of configuration change, such as adding an adapter to a server.

The resource tree on one service may not be consistent across instances of devices. The client should traverse the data model and discover resources to interact with them.

Some resources might remain very stable from system to system, such as manager network settings. However, the architecture does not guarantee this stability.

- A resource tree should remain stable across service restarts and minor device configuration changes. Thus, the set of URIs and array element indexes should remain constant.
- A client shall not expect the resource tree to be consistent between instances of services.

12.4. Discovery

Automatic discovery of managed devices supporting Redfish may be accomplished by using the Simple Service Discovery Protocol (SSDP). This protocol enables network-efficient discovery without resorting to ping-sweeps, router table searches, or restrictive DNS naming schemes. Use of SSDP is optional, and if implemented, shall enable the user to disable the protocol through the ManagerNetworkProtocol resource.

The objective of discovery is for client software to locate managed devices that conform to the Redfish Specification. Therefore, the primary SSDP functionality is incorporated in the M-SEARCH query. Redfish also follows the SSDP extensions and naming that UPnP uses, where applicable, so that systems that conform to the Redfish Specification can also implement UPnP without conflict.
12.4.1. UPnP compatibility

For compatibility with general-purpose SSDP client software, primarily UPnP, the service should use UDP port 1900 for all SSDP traffic. In addition, the Time-to-Live (TTL) hop count setting for SSDP multicast messages should default to 2.

12.4.2. USN format

The UUID in the USN field of the service shall equal the UUID property in the Service Root. If multiple or redundant managers exist, the UUID of the service shall remain static regardless of redundancy failover. The unique ID shall be in the canonical UUID format, followed by ::dmtf-org.

12.4.3. M-SEARCH response

The Redfish Service Search Target (ST) is defined as:

```
urn:dmtf-org:service:redfish-rest:1
```

The managed device shall respond to M-SEARCH queries for Search Target (ST) of the Redfish Service, as well as ssdp:all. For UPnP compatibility, the managed device should respond to M-SEARCH queries for Search Target (ST) of upnp:rootdevice.

The URN provided in the ST header in the reply shall use the redfish-rest: service name followed by the major version of the Redfish Specification. If the minor version of the Redfish Specification to which the service conforms is a non-zero value, that minor version shall be appended with and preceded by a colon (:).

For example, a service that conforms to a Redfish Specification v1.4 would reply with a redfish-rest:1:4 service.

The managed device shall provide clients with the AL header that points to the Redfish Service Root URL.

For UPnP compatibility, the managed device should provide clients with the Location header that points to the UPnP XML descriptor.

The response to an M-SEARCH multicast or unicast query shall use the following format:

```
HTTP/1.1 200 OK
CACHE-CONTROL:max-age=<seconds, at least 1800>
ST:urn:dmtf-org:service:redfish-rest:1
```
A service may provide additional headers for UPnP compatibility. Fields in brackets are placeholders for device-specific values.

12.4.4. Notify, alive, and shutdown messages

Redfish devices may implement the additional UPnP-defined SSDP messages to announce their availability to software. If implemented, services shall allow the end user to disable the traffic separately from the M-SEARCH response functionality. This capability enables users to use the discovery functionality with minimal amounts of generated network traffic.

12.5. Server-Sent Events

12.5.1. General

Server-Sent Events (SSE), as defined by the Web Hypertext Application Technology Working Group, enables a client to open a connection with a web service. The web service can continuously push data to the client, as needed.

Successful resource responses for SSE shall:

- Return the HTTP 200 OK status code.
- Have a Content-Type header set as text/event-stream or text/event-stream; charset=utf-8.

Unsuccessful resource responses for SSE shall:

- Return an HTTP status code of 400 or greater.
- Have a Content-Type header set as application/json or application/json; charset=utf-8.
- Contain a JSON object in the response body, as described in Error responses, which details the error or errors.

A service may occasionally send a comment within a stream to keep the connection alive. Services shall separate events with blank lines. Blank lines should be sent as part of the end of an event, otherwise dispatch may be delayed in conforming consumers.

The following clauses describe how Redfish uses SSE in different Redfish data model contexts. For details about SSE, see the HTML5 Specification.
12.5.2. Event Service

A service's implementation of the EventService resource may contain the ServerSentEventUri property. If a client performs a GET on the URI specified by the ServerSentEventUri property, the service shall keep the connection open and conform to the HTML5 Specification until the client closes the socket. Service-generated events shall be sent to the client by using the open connection.

When a client opens an SSE stream for the Event Service, the service shall create an EventDestination resource in the Subscriptions collection for the Event Service to represent the connection. The Context property in the EventDestination resource shall be a service-generated opaque string.

The service shall delete the corresponding EventDestination resource when the connection is closed. The service shall close the connection if the corresponding EventDestination resource is deleted.

The service shall use the id field in the SSE stream to uniquely identify a payload in the SSE stream. The value of the id field is determined by the service. A service should accept the Last-Event-ID header from the client to allow a client to restart the event stream in case the connection is interrupted.

The service shall use the data field in the SSE stream based on the payload format. The SSE streams have these formats:

- **Metric report SSE stream.** Services shall use this format when the Telemetry Service generates or updates a metric report.
- **Event message SSE stream.** Services shall use this format for all other types of events.

To reduce the amount of data returned to the client, the service should support the $filter query parameter in the URI for the SSE stream.

**Note:** The $filter syntax shall follow the format in the $filter query parameter clause.

The service should support these properties as filter criteria:

- **EventFormatType**

  The service sends events of the matching EventFormatType.

  **Example:**

  ```
  https://sseuri?$filter=EventFormatType eq Event
  ```

  Valid values are the EventFormatType enumerated string values that the Redfish EventService schema defines.
- **EventType**
  
The service sends events of the matching **EventType**.

  **Example:**
  
  ```plaintext
  https://sseuri?$filter=EventType eq StatusChange
  ```

  *Valid values are the **EventType** enumerated string values that the Redfish Event schema defines.*

- **MessageId**
  
The service sends events with the matching **MessageId**.

  **Example:**
  
  ```plaintext
  https://sseuri?$filter=MessageId eq 'Contoso.1.0.TempAssert'
  ```

- **MetricReportDefinition**
  
The service sends metric reports generated from the **MetricReportDefinition**.

  **Example:**
  
  ```plaintext
  ```

- **OriginResource**
  
The service sends events for the resource.

  **Example:**
  
  ```plaintext
  ```

- **RegistryPrefix**
  
The service sends events with messages that are part of the **RegistryPrefix**.
Example:

https://sseuri?$filter=(RegistryPrefix eq Resource) or (RegistryPrefix eq Task)

- ResourceType

The service sends events for resources that match the ResourceType.

Example:

https://sseuri?$filter=(ResourceType eq 'Power') or (ResourceType eq 'Thermal')

- SubordinateResources

When SubordinateResources is true and OriginResource is specified, the service sends events for the resource and its subordinate resources.

Example:

https://sseuri?$filter=(OriginResource eq '/redfish/v1/Systems/1') and (SubordinateResources eq true)

12.5.2.1. Event message SSE stream

The service shall use the data field in the SSE stream to include the JSON representation of the Event object.

The following example payload shows a stream that contains a single event with the id field set to 1, and a data field that contains a single Event object.

```json
id: 1
data: {
data:   "@odata.type": "#Event.v1_1_0.Event",
data:   "Id": "1",
data:   "Name": "Event Array",
data:   "Context": "ABCDEFGH",
data:   "Events": [
data:      }]
```
### 12.5.2.2. Metric report SSE stream

The service shall use the `data` field in the SSE stream to include the JSON representation of the `MetricReport` object.

The following example payload shows a stream that contains a metric report with the `id` field set to 127, and the `data` field containing the metric report object.
12.6. Update Service

12.6.1. Overview

This clause covers the mechanism for software updates by using the Update Service.

12.6.2. Software update types

Clients can use these methods to update software through the Update Service:

- **Simple updates**: The service pulls the update from a client-indicated network location.
- **Multipart HTTP push updates**: The client uses HTTP or HTTPS with a multipart-formatted request body to push a software image to the service.

12.6.2.1. Simple updates

A service may support the SimpleUpdate action within the UpdateService resource. A client can perform a POST to the action target URI to initiate a pull-based update, as define by the UpdateService schema. After a successful POST, the service should return the HTTP 202 Accepted status code with the Location header set to the URI of a task monitor. Clients can use this task to monitor the progress and results of the update, which includes the progress of image transfer to the service.

12.6.2.2. Multipart HTTP push updates

A service may support the MultipartHttpPushUri property within the UpdateService resource. A
client can perform an HTTP or HTTPS POST on the URI specified by this property to initiate a push-based update.

- Access to this URI shall require the same privilege as access to the Update Service.
- A client POST to this URI shall contain the Content-Type HTTP header with the value multipart/form-data, with the body formatted as defined by this specification. For more information about multipart/form-data HTTP requests, see RFC7578.
- The client POST request shall contain the binary image as one of the parts in a multipart/form-data request body, as defined by the following table. In addition, the request shall include parameters for the update in a JSON formatted part in the same multipart/form-data request body, as defined by the following table. If the request has no parameters, an empty JSON object shall be used.
- A service may require the Content-Length HTTP header for POST requests to this URI. In this case, if a client does not include the required Content-Length header in the POST request, the service shall return the HTTP 411 Length Required status code.
- A service should return the HTTP 412 Precondition Failed status code if the size of the binary image is larger than the maximum image size that the service supports, as advertised in MaxImageSizeBytes property in the UpdateService resource.
- After a successful POST to this URI, the service shall return the HTTP 202 Accepted status code with a Location header set to the URI of a task monitor. This task can be used by clients to monitor the progress and results of the update.

The following table describes the requirements of a multipart/form-data request body for HTTP push software update:

<table>
<thead>
<tr>
<th>Request body part</th>
<th>HTTP headers</th>
<th>Header value and parameters</th>
<th>Required</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Update parameters</td>
<td>Content-Disposition</td>
<td>form-data; name=&quot;UpdateParameters&quot;</td>
<td>Yes</td>
<td>JSON-formatted part for passing the update parameters. The value of the name field shall be &quot;UpdateParameters&quot;. The format of the JSON shall follow the definition of the UpdateParameters object in the UpdateService schema.</td>
</tr>
<tr>
<td>JSON part</td>
<td>Content-</td>
<td>application/</td>
<td>Yes</td>
<td>The media type format and character set of this</td>
</tr>
<tr>
<td>Request body part</td>
<td>HTTP headers</td>
<td>Required</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>------------------</td>
<td>--------------</td>
<td>----------</td>
<td>-------------</td>
<td></td>
</tr>
<tr>
<td><strong>Update file binary part</strong></td>
<td>Content-Disposition</td>
<td>Yes</td>
<td>Binary file to use for this software update. The value of the name field shall be &quot;UpdateFile&quot;. The value of the filename field should reflect the name of the file as loaded by the client.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Content-Type</td>
<td>Yes</td>
<td>The media type format of the binary update file.</td>
<td></td>
</tr>
<tr>
<td><strong>OEM specific parts</strong></td>
<td>Content-Disposition</td>
<td>No</td>
<td>Optional OEM part. The value of the name field shall start with &quot;Oem. Content-Type is optional, and depends on the OEM part type.</td>
<td></td>
</tr>
</tbody>
</table>

This example shows a multipart/form-data request to push an update image:

```bash
POST /redfish/v1/UpdateService/upload HTTP/1.1
Host: <host-path>
Content-Type: multipart/form-data; boundary=---------------------------d74496d66958873e
Content-Length: <computed-length>
Connection: keep-alive
X-Auth-Token: <session-auth-token>

-----------------------------d74496d66958873e
Content-Disposition: form-data; name="UpdateParameters"
Content-Type: application/json

{
    "Targets": [/
        /redfish/v1/Managers/1"
    ],
    "@Redfish.OperationApplyTime": "OnReset",
}
```
13. Security details

13.1. Transport Layer Security (TLS) protocol

Implementations shall support the Transport Layer Security (TLS) protocol v1.2 with RFC7525 recommendations or later. Implementations may remove support for older versions for TLS in favor of newer versions.

**DEPRECATED:** Previous versions of this specification allowed for TLS v1.1.

Implementations should support:

- The latest version of the TLS v1.x specification.

13.1.1. Cipher suites

Implementations shall only support cipher suites listed as "Recommended" in the TLS Cipher Suites table defined by the IANA TLS Parameters registry.

Cipher suites that are listed as mandatory in various RFCs, but are not "Recommended" in the TLS Cipher Suites table defined by the IANA TLS Parameters registry, shall not be supported.

Implementations should consider the support of pre-shared key ciphers suites listed as "Recommended" in the TLS Cipher Suites table defined by the IANA TLS Parameters registry, which enable authentication and identification without trusted certificates.

**DEPRECATED**

Implementations should support AES-256-based ciphers from the TLS suites.

Redfish implementations should consider the support of ciphers, such as the following ciphers, which
enable authentication and identification without trusted certificates:

- TLS_PSK_WITH_AES_256_GCM_SHA384
- TLS_DHE_PSK_WITH_AES_256_GCM_SHA384
- TLS_RSA_PSK_WITH_AES_256_GCM_SHA384

The advantage of these recommended ciphers is:

AES-GCM is not only efficient and secure, but hardware implementations can achieve high speeds with low cost and low latency because the mode can be pipelined.

Additionally, Redfish implementations should support the following cipher:

- TLS_RSA_WITH_AES_128_CBC_SHA

For more information, see RFC5288 and RFC5487.

END DEPRECATED

13.1.2. Certificates

Redfish implementations shall support replacement of the default certificate if one is provided.

Redfish implementations shall use certificates that conform to X.509-v3, as defined in RFC5280.

13.2. Operations that contain sensitive data

Operations that contain sensitive data should use HTTPS only. For example, a SimpleUpdate action with a user name and password should use HTTPS to protect the sensitive data.

13.3. Authentication

Services:

- Shall support both HTTP Basic authentication and Redfish session login authentication.
- Shall use only connections that conform to TLS to transport the data between any third-party authentication service and clients.
- Shall not require a client that uses HTTP Basic authentication to create a session.
- May implement other authentication mechanisms.
13.3.1. Authentication requirements

13.3.1.1. Resource and operation authentication requirements

Services shall authenticate all write requests to Redfish resources. For example:

- POST, except to the `Sessions` resource collection for authentication
- PUT
- PATCH
- DELETE

Redfish resources shall not be available as unauthenticated, except for:

- The service root to identify the device and service locations.
- The Redfish metadata document to get resource types.
- The OData service document for compatibility with OData clients.
- The Redfish OpenAPI YAML document for compatibility with OpenAPI clients.
- The `version` object at `/redfish`.

*Note:* This specification does not cover external services that are linked through external references. These services may have other security requirements.

13.3.1.2. HTTP redirect authentication requirements

An HTTP redirect shall enforce the privilege requirements for the target resource.

Generally, if the location is reachable without authentication but only over HTTPS, the service shall issue a redirect to the HTTPS version of the resource. When the resource is accessible only with authentication, the service should return the HTTP `404 Not Found` status code.

13.3.1.3. HTTP header authentication requirements

An authentication header shall accompany every request that establishes a secure channel.

Services:

- Shall process HTTP headers for authentication before other headers that may affect the response. For example, `ETag`, `If-Modified`, and so on.
- Shall not use HTTP cookies to authenticate any activity, such as `GET`, `POST`, `PUT`, `PATCH`, and `DELETE`.

13.3.1.4. Authentication failure requirements

When authentication fails, extended error messages shall not provide privileged information.
13.3.2. HTTP Basic authentication

Services shall support HTTP Basic authentication, as defined by RFC7617, and shall use only connections that conform to TLS to transport the data between any third-party authentication service and clients.

All requests that use HTTP Basic authentication shall require HTTPS.

**Note:** The IETF has highlighted security concerns with HTTP Basic authentication. While HTTPS is required for the usage of HTTP Basic authentication, there are other concerns implementors need to be aware of that are documented in RFC7617.

13.3.3. Redfish session login authentication

Service shall provide login sessions that conform with this specification.

Session management is determined by the implementation of the Redfish Service, which includes orphaned session timeout and the management of the number of simultaneous open sessions.

13.3.3.1. Redfish login sessions

For improved performance and security, a client should use the session management interface to create a Redfish login session. The session service specifies the URI for session management.

To establish a session, find the URI in either:

- The session service's Sessions property.
- The service root's links property under the Sessions property.

Both URIs shall be the same.

```json
{
   "SessionService": {
      "@odata.id": "/redfish/v1/SessionService"
   },
   "Links": {
      "Sessions": {
         "@odata.id": "/redfish/v1/SessionService/Sessions"
      }
   },
   ...
}
```
13.3.3.2. Session login

To create a Redfish session without an authentication header, perform an HTTP POST request to the session service's Sessions resource collection. The POST to create a session shall only be supported with HTTPS. If both HTTP and HTTPS are enabled, a POST request to create a session through the HTTP port should redirect to the HTTPS port. Include the following POST body:

```
POST /redfish/v1/SessionService/Sessions HTTP/1.1
Host: <host-path>
Content-Type: application/json;charset=utf-8
Content-Length: <computed-length>
Accept: application/json;charset=utf-8
OData-Version: 4.0

{
  "UserName": "<username>",
  "Password": "<password>"
}
```

Fields in brackets are placeholders for client-specific values.

To verify that the request has been initiated from an authorized client domain, services should save the Origin header in reference to this session creation and compare it to subsequent requests using this session.

The response to the POST request to create a session shall include:

- **X-Auth-Token header**. Contains a session authentication token that the client can use in subsequent requests.
- **Location header**. Contains a hyperlink to the new Session resource.
- **JSON response body**. Contains the full representation of the new Session resource.

The following is a sample response of a session being created:

```
HTTP/1.1 201 Created
Location: /redfish/v1/SessionService/Sessions/1
X-Auth-Token: <session-auth-token>

{
  "@odata.id": "/redfish/v1/SessionService/Sessions/1",
  "@odata.type": "#Session.v1_0_0.Session",
  "Id": "1",
  "Name": "User Session",
  "Description": "User Session",
```
The client that sends the session login request should save the session authentication token from the `X-Auth-Token` header and the contents of the `Location` header from the response of the login POST request.

To authenticate subsequent requests, the client sets the `X-Auth-Token` header to the session authentication token that the POST login request returns.

**Note:** The session ID differs from the session authentication token, as follows:

- **Session ID:** The session ID uniquely identifies the Session resource. The response data with the last segment of the `Location` header URI returns is the session ID. To view active sessions and terminate any session, an administrator with sufficient privileges can use the session ID.
- **Session authentication token:** Only the client that executes the login has the session authentication token.

### 13.3.3.3. Session lifetime

Unlike some token-based methods that use token expiration times, Redfish sessions time out. As long as a client continues to send requests more frequently than the session timeout period, the session remains open and the session authentication token remains valid. If the session times out, it is automatically terminated.

### 13.3.3.4. Session termination or logout

When the client logs out, the Redfish session terminates. The session terminates through a DELETE request to the `Session` resource defined in either the `Location` header URI or the session ID in the response data.

This ability to DELETE a session through the `Session` resource enables an administrator with sufficient privileges to terminate other users' sessions from a different session.

When a session is terminated, the service shall not affect independent connections established originally by this session for other purposes, such as connections for Server-Sent Events or transferring an image for the update service.
13.4. Authorization

13.4.1. Privilege model

To control which users have access to resources and the type of access that users have, the authorization subsystem uses roles and privileges.

A role is a defined set of privileges. Two roles with the same privileges shall behave equivalently. Each user shall be assigned exactly one role.

This Specification defines a set of predefined roles, where Role name is the value of the Id property in the Role resource:

<table>
<thead>
<tr>
<th>Role name</th>
<th>Assigned privileges</th>
</tr>
</thead>
<tbody>
<tr>
<td>Administrator</td>
<td>Login, ConfigureManager, ConfigureUsers, ConfigureComponents, ConfigureSelf</td>
</tr>
<tr>
<td>Operator</td>
<td>Login, ConfigureComponents, ConfigureSelf</td>
</tr>
<tr>
<td>ReadOnly</td>
<td>Login, ConfigureSelf</td>
</tr>
</tbody>
</table>

Services shall support the previous predefined roles. Services may include OEM privileges in the predefined roles. The AssignedPrivileges property in the Role resource for the predefined roles shall not be modifiable.

A service may define additional custom roles. A service may allow users to create custom roles through a POST to the Roles resource collection.

A predefined role or a custom role shall be assigned to a user when a manager account is created. The client shall provide the RoleId property when creating a manager account to select the predefined role or a custom role.

A privilege is a permission to complete an operation, such as read or write, within a defined management domain, such as configuring users. The AssignedPrivileges array in the Role resource defines a set of assigned privileges for the associated role. A service may also include OemPrivileges in the Role resource for additional privileges.

Services shall enforce the same privilege model for ETag-related activity as is enforced for the data being represented by the ETag. For example, when the activity that requires privileged access to read a data item that the ETag represents requires the same privileged access to read the ETag.
13.4.2. Redfish service operation-to-privilege mapping

For every request that a client makes to a service, the service shall determine that the authenticated
identity of the requester has the authorization to complete the requested operation on the resource in the
request.

Using the role and privileges authorization model where an authenticated identity context is assigned a
role and a role is a set of privileges, the service typically checks an HTTP request against a mapping of
the authenticated requesting identity role and privileges to determine whether the identity privileges are
sufficient to complete the operation in the request.

13.4.2.1. Why specify operation-to-privilege mapping?

Initial versions of the Redfish Specifications defined several role-to-privilege mappings for standardized
roles and normatively identified several privilege labels but did not normatively detail what these
privileges or how privilege-to-operations mappings could be specified or represented in a normative
fashion.

The lack of a methodology to define which privileges are required to complete a requested operation
against the URI in the request puts at risk the interoperability between service implementations that
clients may encounter due to variances in privilege requirements between implementations.

Also, a lack of methodology for specifying and representing the operation-to-privilege mapping prevents
the Redfish Forum or other governing organizations from normatively defining privilege requirements for a
service.

13.4.2.2. Representing operation-to-privilege mappings

A service should provide a Privilege Registry in the registry collection. This registry represents the
privileges required to complete HTTP operations against resources supported by the service.

The Privilege Registry is a JSON document that contains a `Mappings` array of where an individual entry
exists for every resource type that the service supports.

The operation-to-privilege mapping is defined for every resource type and applies to every resource the
service implements for the applicable resource type.

In several situations, specific resources or properties may have differing operation-to-privilege mappings
than the resource type-level mappings. In these cases, the resource type-level mappings need to be
overrides. The `PrivilegeRegistry` schema defines the methodology for resource type-level
operation-to-privilege mappings and related overrides.

If a service provides a Privilege Registry, the service shall use the Redfish Forum's Privilege Registry
definition as a base operation-to-privilege mapping definition for operations that the service supports to
promote interoperability for Redfish clients.
13.4.2.3. Operation map syntax

An operation map defines the set of privileges required to complete an operation on a resource-type.

The mapped operations are GET, PUT, PATCH, POST, DELETE, and HEAD. A privilege mapping is defined for each operation, irrespective of whether the service or data model supports the operation on the resource-type.

The privilege labels may be the Redfish standardized labels that the PrivilegeType enumeration in the Privileges schema defines and they may be OEM-defined privilege labels. The required privileges for an operation are specified using logical AND and OR behavior. For more information, see the Privilege AND and OR syntax clause.

The following example defines the privileges required for various operations on the Manager resource. Unless the implementation defines mapping overrides to the OperationMap array, the specified operation-to-privilege mapping represents behavior for all Manager resources in a service implementation.

```json
{
    "Entity": "Manager",
    "OperationMap": {
        "GET": [
            {
                "Privilege": [ "Login" ]
            }
        ],
        "HEAD": [
            {
                "Privilege": [ "Login" ]
            }
        ],
        "PATCH": [
            {
                "Privilege": [ "ConfigureManager" ]
            }
        ],
        "POST": [
            {
                "Privilege": [ "ConfigureManager" ]
            }
        ],
        "PUT": [
            {
                "Privilege": [ "ConfigureManager" ]
            }
        ]
    }
}
```
13.4.2.4. Mapping overrides syntax

In several situations, operation-to-privilege mapping varies from the resource type-level mapping.

<table>
<thead>
<tr>
<th>Situation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Property override</td>
<td>A property has different privilege requirements than the resource in which it resides. For example, the Password property in the ManagerAccount resource requires the ConfigureSelf or ConfigureUsers privilege to change, in contrast to the ConfigureUsers privilege required for the other properties in ManagerAccount resources. If multiple properties with the same name are present in a resource, the property override applies to all property instances.</td>
</tr>
<tr>
<td>Subordinate override</td>
<td>A resource is used in context of another resource and the contextual privileges need to govern. For example, the privileges for PATCH operations on EthernetInterface resources depend on whether the resource is subordinate to the Manager resource, where ConfigureManager is required, or the ComputerSystem resource, where ConfigureComponents is required.</td>
</tr>
<tr>
<td>Resource URI override</td>
<td>A resource instance has different privilege requirements for an operation than those defined for the resource type.</td>
</tr>
</tbody>
</table>

The overrides are defined in the context of the operation-to-privilege mapping for a resource type.

If multiple overrides are specified for a single resource type, the following precedence should be used for determining the appropriate override to apply:

- Property override
- Resource URI override
- Subordinate override

13.4.2.5. Property override example

In the following example, the Password property on the ManagerAccount resource requires the
ConfigureSelf or ConfigureUsers privilege to change, in contrast to the ConfigureUsers privilege required for the other properties in ManagerAccount resources:

```json
{
    "Entity": "ManagerAccount",
    "OperationMap": {
        "GET": [
            {
                "Privilege": [ "ConfigureManager" ]
            },
            {
                "Privilege": [ "ConfigureUsers" ]
            },
            {
                "Privilege": [ "ConfigureSelf" ]
            }
        ],
        "HEAD": [
            {
                "Privilege": [ "Login" ]
            }
        ],
        "PATCH": [
            {
                "Privilege": [ "ConfigureUsers" ]
            }
        ],
        "POST": [
            {
                "Privilege": [ "ConfigureUsers" ]
            }
        ],
        "PUT": [
            {
                "Privilege": [ "ConfigureUsers" ]
            }
        ],
        "DELETE": [
            {
                "Privilege": [ "ConfigureUsers" ]
            }
        ]
    },
    "PropertyOverrides": {
        "Targets": [ "Password" ],
        "OperationMap": {
```
13.4.2.6. Subordinate override

The **Targets** property in **SubordinateOverrides** lists a hierarchical representation for when to apply the override. In the following example, the override for an **EthernetInterface** resource is applied when it is subordinate to an **EthernetInterfaceCollection** resource, which in turn is subordinate to a **Manager** resource. If a client were to PATCH an **EthernetInterface** resource that matches this override condition, it requires the **ConfigureManager** privilege. Otherwise, the client requires the **ConfigureComponents** privilege.
13.4.2.7. Resource URI override

The following example demonstrates the resource URI override syntax to define operation privilege
variations for resource URIs.

The example defines both `ConfigureComponents` and `OEMAdminPriv` privileges as required to make a PATCH operation on the two resource URIs listed as targets.

```json
{
    "Entity": "ComputerSystem",
    "OperationMap": {
        "GET": [
            {
                "Privilege": [ "Login" ]
            }
        ],
        "HEAD": [
            {
                "Privilege": [ "Login" ]
            }
        ],
        "PATCH": [
            {
                "Privilege": [ "ConfigureComponents" ]
            }
        ],
        "POST": [
            {
                "Privilege": [ "ConfigureComponents" ]
            }
        ],
        "PUT": [
            {
                "Privilege": [ "ConfigureComponents" ]
            }
        ],
        "DELETE": [
            {
                "Privilege": [ "ConfigureComponents" ]
            }
        ]
    }
},
"ResourceURIOverrides": [
    {
        "Targets": [
            "/redfish/v1/Systems/VM6",
            "/redfish/v1/Systems/Sys1"
        ],
        "OperationMap": {
            "GET": [
```
13.4.2.8. Privilege AND and OR syntax

The array placement of the privilege labels in the OperationMap GET, HEAD, PATCH, POST, PUT, and DELETE operation element arrays define the logical combinations of privileges that are required to call an operation on a resource or property.

For OR logical combinations, the privilege label appears in the operation element array as individual elements.

The following example defines either Login or OEMPrivilege1 privileges as required to perform a GET operation.

```json
{
    "GET": [
        {
            "Privilege": [ "Login" ]
        },
        {
            "Privilege": [ "OEMPrivilege1" ]
        }
    ]
}
```

For logical AND combinations, the privilege label appears in the Privilege property array in the operation element.

The following example defines both ConfigureComponents and OEMSysAdminPriv as required to perform a PATCH operation.

```json
{
    "PATCH": [
        {
            "Privilege": [ "ConfigureComponents","OEMSysAdminPriv" ]
        }
    ]
}
```
13.5. Account service

- Implementations should store user passwords with one-way encryption techniques.
- Implementations may support exporting user accounts with passwords, but shall do so using encryption methods to protect them.
- User accounts shall support ETags and atomic operations. Implementations may reject requests that do not include an ETag.
- When authentication fails, extended error messages shall not provide privileged information.

13.5.1. Password management

A Redfish service provides local user accounts through a collection of ManagerAccount resources located under the account service. The ManagerAccount resources enable users to manage their own account information, and for administrators to create, delete, and manage other user accounts.

When account properties are changed, the service may close open sessions for this account and require re-authentication.

13.5.2. Password change required handling

The service may require that passwords assigned by the manufacturer be changed by the end user prior to accessing the service. In addition, administrators may require users to change their account's password upon first access.

The ManagerAccount resource contains a PasswordChangeRequired boolean property to enable this functionality. Resources that have the property set to true shall require the user to change the write-only Password property in that resource before access is granted. Manufacturers including user credentials for the service may use this method to force a change to those credentials before access is granted.

When a client accesses the service by using credentials from a ManagerAccount resource that has a PasswordChangeRequired value of true, the service shall allow:

- A session login and include a @Message.ExtendedInfo object in the response containing the
PasswordChangeRequired message from the Base Message Registry. This indicates to the client that their session is restricted to performing only the password change operation before access is granted.

- A GET operation on the ManagerAccount resource associated with the account.
- A PATCH operation on the ManagerAccount resource associated with the account to update the Password property. If the value of Password is changed, the service shall also set the PasswordChangeRequired property to false.

For all other operations, the service shall respond with the HTTP 403 Forbidden status code and include a @Message.ExtendedInfo object that contains the PasswordChangeRequired message from the Base Message Registry.

13.6. Asynchronous tasks

Irrespective of which user or privileged context starts a task, the information in the task object shall enforce the privileges required to access that object.

13.7. Event subscriptions

Before pushing event data object to the destination, the service may verify the destination for identity purposes.

14. Redfish Host Interface

The Redfish Host Interface Specification defines how software that runs on a host computer system can interface with a Redfish Service that manages the host. For details, see DSP0270.

15. Redfish Composability

A service may implement the CompositionService resource off of ServiceRoot to bind resources. One example is disaggregated hardware, which allows for independent components, such as processors, memory, I/O controllers, and drives, to be bound together to create logical constructs that operate together. This enables a client to dynamically assign resources for an application.

A service that supports composability shall implement Resource Blocks, defined by the ResourceBlock schema, and the Resource Zones, defined in the Zone schema, for the Composition Service. Resource Blocks provide an inventory of components available to the client for building compositions. Resource Zones describe the binding restrictions of the Resource Blocks that the service manages.

The Resource Zones within the Composition Service shall include the collection capabilities annotation in...
responses. The collection capabilities annotation allows a client to discover which resource collections in
the service support compositions, the different composition request types allowed, and how the POST
request for the resource collection is formatted, as well as what properties are required.

15.1. Composition requests

A service that implements the Composition Service, as defined by the CompositionService schema,
shall support one or more of the following types of composition requests:

- Specific composition
- Constrained composition
- Expandable resources

A service that supports the removal of a composed resource shall support the DELETE method on the
composed resource.

15.1.1. Specific composition

A specific composition is when a client identifies an exact set of resources in which to build a logical
entity.

A service that supports specific compositions shall support a POST request that contains an array of
hyperlinks to Resource Blocks. The schema for the resource being composed defines where the
Resource Blocks are specified in the request.

The following example shows a ComputerSystem being composed with a specific composition request:

```
POST /redfish/v1/Systems HTTP/1.1
Content-Type: application/json;charset=utf-8
Content-Length: <computed length>
OData-Version: 4.0

{
  "Name": "Sample Composed System",
  "Links": {
    "ResourceBlocks": [
      {
        "@odata.id": "/redfish/v1/CompositionService/ResourceBlocks/ComputeBlock0"
      },
      {
        "@odata.id": "/redfish/v1/CompositionService/ResourceBlocks/DriveBlock2"
      }
    ]
  }
}
```
15.1.2. Constrained composition

A constrained composition is when a client has identified a set of criteria, or constraints, in which to build a logical entity. This includes criteria such as quantities of components, or characteristics of components. A service that supports constrained compositions shall support a POST request that contains the set of characteristics to apply to the composed resource. The specific format of the request is defined by the schema for the resource being composed. This type of request may include expanded elements of resources subordinate to the composed resource.

The following example shows a ComputerSystem being composed with a constrained composition request:

```
POST /redfish/v1/Systems HTTP/1.1
Content-Type: application/json;charset=utf-8
Content-Length: <computed length>
OData-Version: 4.0

{
    "Name": "Sample Composed System",
    "PowerState": "On",
    "BiosVersion": "P79 v1.00 (09/20/2013)",
    "Processors": {
        "Members": [
            {
                "@Redfish.RequestedCount": 4,
                "@Redfish.AllowOverprovisioning": true,
                "ProcessorType": "CPU",
                "ProcessorArchitecture": "x86",
                "InstructionSet": "x86-64",
                "MaxSpeedMHz": 3700,
                "TotalCores": 8,
                "TotalThreads": 16
            }
        ]
    },
    "Memory": {
        "Members": [
```
15.1.3. Expandable resources

An expandable resource is when a service has a baseline composition that cannot be removed. Instead of a client making requests to create a composed resource, a client can only add or remove resources from the composed resource. A service that supports expandable resources shall support one or more of
the update methods that the Updating a composed resource clause describes.

15.2. Updating a composed resource

A service that supports updating a composed resource shall provide one or more of the following methods to update composed resources:

- The PUT or PATCH methods on the composed resource with a modified list of Resource Blocks.
- Actions on the composed resource for adding and removing Resource Blocks.
  - If the actions for adding and removing Resource Blocks are present in the resource, clients should use this method before attempting PUT or PATCH.

16. Aggregation

Aggregation has been a Redfish concept since its inception. Redfish uses collection for services that can represent more than one system. As the scale of Redfish implementations increase, there is a desire for clients to operate on Redfish resources en masse.

Aggregation is the representation of Redfish resources from a variety of sources so that they can be managed, in whole or in part, by a Redfish client. Membership can be heterogeneous and arbitrary, but it is expected that most aggregate members will be of the same resource type, such as an aggregation of ComputerSystem resource, which would be represented by an Aggregate resource whose members of its Elements array are exclusively of type ComputerSystem. The Redfish service proxies on behalf of the aggregated components in order to provide common operations. The Redfish service is representing resources on behalf of the components and incoming operations must be tracked by the Redfish service before being accomplished by communicating with the individual resources. Thus, aggregation also allows a Redfish client to act on resources as a group using aggregates.

16.1. Classes of aggregators

There are at least two classes of Redfish aggregators.

There are implicit aggregators. An example of this might be an enclosure manager, such as a manager of blades in an enclosure. This implementation has ComputerSystem resources representing blades in the ComputerSystemCollection resource, and one or more Manager resources in the ManagerCollection resource. It also would likely have a Chassis resource for each blade and a Chassis resource for the enclosure, which would use the Contains property in Links to express the containment relationship to the individual blades. This class of aggregator has been in existence since Redfish inception as it will proxy to/from blades to perform management functions.

There are complex aggregators. An example of this type of aggregator might be a rack level manager, fabric manager, or a manager of similar scale, especially if it is representing resources it gathers by
proxying information from other managers like BMCs. The sources that this manager aggregates are more complex in nature and potentially varying. This manager probably has an interface to the resources and is proxying the Redfish service on behalf of each set of resources. At this scale, a Redfish client would prefer to provide common functions, such as resetting a set of systems, to the Redfish service as a whole rather than invoking actions individually in order to achieve scalability requirements. This class of service also may need assistance in adding members to the service, such as providing address and account information in order for the aggregator to contact the components and initiate proxying Redfish operations.

16.1.1. Use cases

There are several use cases that make explicit aggregator representation necessary. What they have in common is the need for common functions for scalability. There are several classes of these common functions.

First there are the service type of common function. An example of this would be firmware update on a large number of systems. Rather than invoke actions on individual resources, it is more efficient for a client to indicate the image and the list of resources to which it is applied. In this case, there is already a service in existence in the model and so an aggregation service isn't needed and instead the existing service needs just be augmented to enable the application to a list of resources.

And additional use case is existing common actions. Examples of this are the ComputerSystem actions Reset or SetDefaultBootOrder. These actions exist, but the Redfish URI structure requires the action to be on each resource. Thus there is an individual operation on each resource. It is more efficient for a client to, instead, provide the action and a list of the resources to which the action needs to be applied. Think of rebooting a thousand systems. One operation with a list of those thousand systems is less overhead for the client as opposed to doing a thousand operations.

A final use case would be to change an attribute on multiple members of a collection. An example of this might be to change the boot order on a large number of systems. Right now, this would require an operation per systems. Provided the resources are in the same collection, the deep PATH operation suffices to meet the requirements of this use case.

16.2. Aggregation service

The AggregationService resource is used to represent that this Redfish service provides aggregation functions. The aggregation service is used to house the group actions that can take place on groups of resources. It also has a resource collection in it for Aggregate resources. Aggregates are the grouping mechanism used by clients to indicate to the service that this group of resources can be treated the same for certain functions, such as the actions.

Actions that are enabled by the service appear within the resource. The list of available actions are schema defined and available in the AggregationService schema, thus there should be a degree of
interoperability guaranteed.

Actions in the AggregationService resource take an argument that is an array of resources as one of the parameters. In many cases, the parameters will include the URIs of individual resources to which the action will apply. Actions in the Aggregate resource may take an argument that is an array of resources as one of the parameters. In this case, any parameters that include the URIs of individual resources to which the action will apply shall only allowed to be that of elements of the aggregate by the service.

Actions that are supported on an Aggregate resource but not supported on all members of the Elements array, such as a Reset action that is not supported on an individual Elements array member, are not silently skipped. If all of the members of the Elements array do not support the method, then an HTTP 4XX status code shall be returned and the body shall contain an error response. If at least one of member of the Elements array successfully completed the action but others did not, the status code should be a 200_OK in this case with @Message.ExtendedInfo objects for the failed elements.

Additionally there are action to add and remove elements from an Aggregate resource. While the Elements array of an Aggregate resource can be PATCHed, the actions make the complexities of dealing with large arrays of objects easier for the Redfish client.

16.2.1. Aggregator requirements

A complex aggregator shall represent it is capable of the following requirements by implementing the AggregationService resource and including an AggregationSourceCollection resource. If there is an AggregationService resource with an AggregationSourceCollection resource, the Redfish service shall:

- Proxy to the aggregated resources on behalf of the service.
- Provide error/state propagation, such as health roll-up, when needed to provide such data to the parent resource.
- Combine resource collections from the aggregated resources.
  - For example, all of any proxied ComputerSystem resources shall be in one ComputerSystemCollection resource.
  - This also implies "URI fix-up" for Redfish provider resources since every system can't be at /redfish/v1/Systems/1.
  - It is advisable for Redfish implementations to use unique values for the Id properties. For example, base the Id of a ComputerSystem resource on something unique like a UUID or serial number, manufacturer MAC address for network adapters, or WWN for Fibre Channel controllers.
- Unify other services.
  - There is only one event service hosted by the aggregation implementation and it shall combine all of the events into one stream. This is also true for sessions, telemetry, update service, and other services. Thus the aggregator is representing a unification of Redfish services that it communicates with and proxies on the client's behalf to the
If there is an `AggregationService` resource, the Redfish service can:

- Support `Aggregate` resources through the `AggregateCollection`.
- Support `AggregationSource` resources by implementing the `AggregationSourceCollection`.
- Support `ConnectionMethod` resources by implementing the `ConnectionMethodCollection`.

### 16.2.2. Aggregates

The `Aggregate` resource is the more formally defined mechanism for grouping resources. Each aggregate contains the list of individual resources that are to be treated as a single unit for operations. Thus if a client wishes to, for example, express that a subset of the `ComputerSystemCollection` resource be treated as a single unit for certain operations like a `Reset`, `SetDefaultBootOrder` action, or firmware update it can express the `Aggregate` resource as the target URI for the operation.

Support of the `AggregateCollection` resource is optional.

### 16.2.3. Aggregation sources and connection methods

The aggregation service model also includes a definition for the information used to access the resources being represented by the aggregator. Two collections of resources are used to represent this. These are the `AggregationSource` and `ConnectionMethod` resources.

The `AggregationSource` resource is used to represent connection to the nexus of information for the resources being reflected by the aggregator. It contains information needed to access that nexus such as the address and account information needed. It also has a reference to the `ConnectionMethod` resource used to access it.

The `ConnectionMethod` resource is used to represent the protocol and other semantics required to communicate with the resources being aggregated. Examples of connection methods are Redfish, IPMI, and proprietary access methods. For methods such as IPMI, it's possible to express the variations and nuances from multiple vendors in this resource.

Support of the `AggregationSourceCollection` and `ConnectionMethodCollection` resources is optional.
17. ANNEX A (informative)

17.1. Change log

<table>
<thead>
<tr>
<th>Version</th>
<th>Date</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.11.0</td>
<td>2020-04-30</td>
<td>Added <strong>Aggregation</strong> clause.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Clarified that services are allowed use HTTP 501 Not Implemented for unsupported HTTP methods.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Clarified the normative semantics around the term &quot;deprecated&quot;.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Clarified clauses describing the usage of null for properties versus not reporting a property.</td>
</tr>
<tr>
<td>1.10.0</td>
<td>2020-03-27</td>
<td>Restructured the <strong>Security details</strong> clause for ease of reading. Other than the changes listed below, no other changes were intended. Any clarifications that inadvertently altered the normative behavior are considered errata, and will be corrected in future revisions to the specification.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Deprecated TLS v1.1, and set the minimum TLS requirement to be TLS v1.2 with RFC7525 recommendations.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Deprecated existing cipher suites clause in favor of new clause to leverage IANA recommendations.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Added requirement for supporting the URI /redfish.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Added support for deep operations.</td>
</tr>
<tr>
<td>1.9.1</td>
<td>2020-03-27</td>
<td>Deprecated full ISO8601 duration format in favor of a simplified version that does not contain years, months, and weeks.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Added missing normative language for how actions with response bodies are defined in schema.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Added HTTP 201 Created as valid responses for actions.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Clarified the ~ operator for the $expand query parameter to expand hyperlinks found in all Links properties.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Clarified the * and . operators for the $expand query parameter to expand hyperlinks found in payload annotations, such as</td>
</tr>
<tr>
<td>Version</td>
<td>Date</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>@Redfish.Settings.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Clarified usage of action parameters that point to resources; the expectation is a reference object pointing to the resource in question is passed by the client.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Clarified that DELETE on a resource will likely delete subordinate resources.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Clarified best practices for naming rules, in particular with regards to acronyms.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Clarified behavior for when individual members of a resource collection cannot be returned as part of a $expand request.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Clarified usage of @Message.ExtendedInfo in error responses and provided guidance for clients for handling error responses.</td>
</tr>
<tr>
<td>1.9.0</td>
<td>2019-12-06</td>
<td>Made change to no longer require the Server response header.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Added clause to Schema modification rules to allow for the addition of OEM URIs to standard resources.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Loosened requirements on @odata.type within Oem to not require it in arrays where the type is used repeatedly.</td>
</tr>
<tr>
<td>1.8.1</td>
<td>2019-12-06</td>
<td>Many changes for style consistency, grammar, and general clarity. Except for the following additions, no normative changes were made. Any clarifications that inadvertently altered the normative behavior are considered errata, and will be corrected in future revisions to the Specification.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Clarified SSE with regards to requiring a blank line after each event.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Clarified order of precedence for resolving multiple operation overrides within the Privilege Registry.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Clarified cases for property overrides in the Privilege Registry where multiple objects in the same resource contain the same property name.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Updated references for HTTP Basic authentication to use RFC7617 instead of RFC7235.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Added text/event-stream, application/yaml, and application/vnd.oai.openapi usage to the Accept and</td>
</tr>
<tr>
<td>Version</td>
<td>Date</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>----------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Content-Type</strong> header table entries.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Added clause that provides guidance on service behavior when null is a property value in POST (create) operations.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Loosened requirements on SSE id based on client usage.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Added documentation for settings, settings apply time, operation apply time, operation apply time support, maintenance window, collection capabilities, requested count, allow over-provisioning, zone affinity, supported certificates, and deprecated terms to the <strong>Payload annotations</strong> clause.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Added clauses that document responses for actions with a response body defined in schema.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Clarified the allowable values payload annotation to show it can be used for both properties and action parameters.</td>
</tr>
<tr>
<td>1.8.0</td>
<td>2019-08-08</td>
<td>Added clause for using /redfish/v1/openapi.yaml as the well-known URI for the OpenAPI document.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Added clause that specifies non-resource reference properties with Uri in the name are accessed using Redfish protocol semantics.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Added <strong>SubordinateResources $filter</strong> parameter for SSE.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Added <strong>Update Service</strong> clause that describes requirements for the <strong>SimpleUpdate</strong> action and the <strong>MultipartHttpPushUri</strong> property.</td>
</tr>
<tr>
<td>1.7.1</td>
<td>2019-08-08</td>
<td>Added statements about the <strong>owning entity</strong> annotation term and its usage in schema modifications.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Clarified SSE id from Id in an event payload and EventId within an event record.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Fixed recommended sequencing of the SSE id to be related to EventId within an event record.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Clarified that services are allowed to close sessions for an account when its password has changed.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Corrected the <strong>Password management</strong> clause to describe how a user can GET their respective account resources when a password change</td>
</tr>
<tr>
<td>Version</td>
<td>Date</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>-----------</td>
<td>-------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>is required.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Clarified that registries are not required to return <code>@odata.id</code>.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Clarified that services should use HTTP 400 Bad Request for invalid query requests.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Clarified that services should use HTTP 400 Bad Request when the only query is being combined with other query parameters.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Clarified that services should use HTTP 400 Bad Request when query parameters are used on non-GET operations.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Added clause about how to construct enumeration values.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Clarified references to specific messages to also reference their Message Registry.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Added language about the construction of action names in payloads.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Added informative text for how OEM actions can be defined.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Added guidance for using HTTPS whenever sensitive data is being transmitted.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Added clause restricting the maximum size of an event payload to be 1MiB.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Clarified that auto expanded resource collections can use paging.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Clarified error response format for SSE.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Clarified that <code>charset=utf-8</code> is not required within the <code>Content-Type</code> header for SSE.</td>
</tr>
<tr>
<td>1.7.0</td>
<td>2019-05-16</td>
<td>The Specification has been significantly rewritten for clarity. Except for the following additions, no normative changes were made. Any clarifications that inadvertently altered the normative behavior are considered errata, and will be corrected in future revisions to the Specification.</td>
</tr>
<tr>
<td>Version</td>
<td>Date</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Added normative statements about how to handle array properties and PATCH operations on arrays.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Separated data model and schema language clauses.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Added clauses that describe how JSON Schema and OpenAPI files are formatted.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Added clause that describes the schema versioning methodology.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Added clause about how URI patterns are constructed based on the resource tree and property hierarchy.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Added Dictionary file naming rules and repository locations.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Enhanced localization definitions and defined repository locations.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Added statement about SSE to the <strong>Eventing mechanism</strong> clause.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Added <strong>Constrained composition</strong> and <strong>Expandable resources</strong> clauses to Redfish Composability.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Added clause about requiring event subscriptions to be persistent across service restarts.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Added clause about persistence of tasks generated as a result of using @Redfish.OperationApplyTime across service restarts.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Added clause about using @Redfish.OperationApplyTime and @Redfish.MaintenanceWindow within task responses.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Removed @odata.context property from example payloads.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Added <strong>Password management</strong> clause to describe functional behavior for restricting access when an account requires a password change.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Added clause around the usage of the HTTP 403 Forbidden status code when an account requires a password change.</td>
</tr>
<tr>
<td>1.6.1</td>
<td>2018-12-13</td>
<td>Added clause about percent encoding being allowed for query parameters.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Changed $expand example to use SoftwareInventory instead of LogEntry.</td>
</tr>
<tr>
<td>Version</td>
<td>Date</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Added clause about the use of a separator for multiple query parameters.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Fixed <code>$filter</code> examples to use <code>/</code> instead of <code>.</code> for property paths.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Clarified the usage of messages in a successful action response; provided an example.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Added clarification about services supporting a subset of HTTP operations on resources specified in schema.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Added clarification about services implementing writable properties as read only.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Added clarification about session termination not affecting connections opened by the session.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Added &quot;Redfish Provider&quot; term definition.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Updated JSON Schema references to point to Draft 7 of the JSON Schema Specification.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Added clarifications about scenarios for when a request to add an event subscription contains conflicting information and how services respond.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Removed language about ignoring the <code>Links</code> property in PATCH requests.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Clarified usage of ETags to show that a client is not supposed to PATCH <code>@odata.etag</code> when attempting to use ETag protection for a resource.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Clarified usage of the <code>only</code> query parameter to show it's not to be combined with <code>$expand</code> and not to be used with singular resources.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Clarified the usage of the HTTP status codes with task monitors.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Various spelling and grammar fixes.</td>
</tr>
<tr>
<td>1.6.0</td>
<td>2018-08-23</td>
<td>Added methods of using <code>$filter</code> on the SSE URI for the Event Service.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Added support for the OpenAPI Specification v3.0. This allows OpenAPI-conforming software to access Redfish Service implementations.</td>
</tr>
<tr>
<td>Version</td>
<td>Date</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Added strict definitions for the URI patterns used for Redfish resources to support OpenAPI. Each URI is now constructed using a combination of fixed, defined path segments and the values of Id properties for resource collections. Also added restrictions on usage of unsafe characters in URIs. Implementations reporting support for Redfish v1.6.0 conform to these URI patterns.</td>
</tr>
<tr>
<td>Version</td>
<td>Date</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>------</td>
<td>-------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>derived from <code>ReferenceableMembers</code>.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Reorganized <strong>Eventing</strong> clause to break out the different subscription methods to differentiate pub-sub from SSE.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Removed statements referencing OData conformance levels.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Clarified terminology to explain usage of absolute versus relative reference throughout.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Clarified client-side HTTP <code>Accept</code> header requirements.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Added evaluation order for supported query parameters and clarified examples.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Clarified handling of annotations in response payloads when used with <code>$select</code> queries.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Clarified service handling of annotations in PATCH requests.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Clarified handling of various PATCH request error conditions.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Clarified ability to create resource collection members by POST operations to the resource collection or the <code>Members</code> array within the resource.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Corrected several examples to show required properties in payload.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Clarified usage of the <code>Link</code> header and values of <code>rel=describedBy</code>.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Clarified that the HTTP status code table only describes Redfish-specific behavior and that unless specified, all other usage follows the definitions within the appropriate RFCs.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Added entry for the HTTP 431 Request Header Fields Too Large status code.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Added statement that the HTTP 503 Service Unavailable status code can be used during reboot or reset of a service to indicate that the service is temporarily unavailable.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Clarified usage of the <code>@odata.type</code> annotation within embedded objects.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Added statements about the required <code>Name</code>, <code>Id</code>, and <code>MemberId</code>.</td>
</tr>
<tr>
<td>Version</td>
<td>Date</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>properties, and the common <strong>Description</strong> property, which have always been shown as required in schema files, but which the Specification did not mention.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Added guidance for the value of time-date properties when time is unknown.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Added the <strong>title</strong> property description in actions.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Clarified usage of the <code>@odata.nextLink</code> annotation at the end of resource collections.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Added additional guidance for naming properties and enumeration values that contain &quot;OEM&quot; or that include acronyms.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Corrected requirements for description and long description annotations.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Corrected name of ConfigureComponents in the Operation-to-privilege mapping clause.</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Various typographical errors and grammatical improvements.</td>
</tr>
<tr>
<td>1.5.0</td>
<td>2018-04-05</td>
<td>Added support for Server-Sent Eventing for streaming events to web-based GUIs or other clients.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Added <code>@Redfish.OperationApplyTime</code> annotation to provide a mechanism for specifying deterministic behavior for the application of Create, Delete or Action (POST) operations.</td>
</tr>
<tr>
<td>1.4.1</td>
<td>2018-04-05</td>
<td>Updated name of the DMTF Forum from SPMF to Redfish Forum.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Consistently used the term, <strong>hyperlink</strong>.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Added example to clarify usage of <code>$select</code> query parameter with <code>$expand</code>, and clarified expected results when using <code>AutoExpand</code>. Corrected order of precedence for <code>$filter</code> parameter options.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Corrected terminology for OEM-defined actions removing &quot;custom&quot; in favor of OEM, and clarified that the action <code>target</code> property is always required for an action, along with its usage.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Corrected location header values for responses to data modification requests that create a task (<code>Task.resource</code> vs. task monitor). Clarified</td>
</tr>
<tr>
<td>Version</td>
<td>Date</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>----------</td>
<td>-------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>error handling of DELETE operations on Task resources.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Removed references to obsolete and unused Privilege annotation namespace.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Clarified usage of the Base.1.0.GeneralError message in the Base Message Registry.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Added durable URIs for registries and profiles, and clarified intended usage for each folder in the repository. Added file naming conventions for registries and profiles, and clarified file naming for schemas.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Added statement to clarify that additional headers may be added to M-SEARCH responses for SSDP to enable UPnP compatibility.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Clarified assignment requirements for predefined or custom roles when new manager account instances are created, using the RoleId property.</td>
</tr>
<tr>
<td>1.4.0</td>
<td>2017-11-17</td>
<td>Added support for optional query parameters ($expand, $filter, and $select) on requests to enable more efficient retrieval of resources or properties from a Redfish Service.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Clarified HTTP status and payload responses after successful processing of data modification requests. This includes POST operations for performing actions, as well as other POST, PATCH, or PUT requests.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Added entries for the HTTP 428 Precondition Required and 507 Insufficient Storage status codes to clarify the proper response to certain error conditions. Added reference links to the HTTP status code table throughout.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Updated the Abstract to reflect the current state of the Specification.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Added reference to RFC6585 and clarified expected behavior when ETag support is used in conjunction with PUT or PATCH operations.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Added definition for &quot;Property&quot; term and updated text to use term consistently.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Added &quot;Client requirement&quot; column and information for HTTP headers on requests.</td>
</tr>
<tr>
<td>Version</td>
<td>Date</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Clarified the usage and expected format of the @odata.context property value.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Added clause to describe how to revise structured properties and resolve their definitions in schema.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Added more descriptive definition for the settings resource. Added an example for the SettingsObject. Added description and example for using the @Redfish.SettingsApplyTime annotation.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Added Action example using the ActionInfo resource in addition to the simple @Redfish.AllowableValues example. Updated example to show a proper subset of the available enumerations to reflect a real-world example.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Added statement explaining the updates required to TaskState upon task completion.</td>
</tr>
<tr>
<td>1.3.0</td>
<td>2017-08-11</td>
<td>Added support for a service to optionally reject a PATCH or PUT operation if the If-Match or If-Match-None HTTP header is required by returning the HTTP 428 Precondition Required status code.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Added support for a service to describe when the values in the settings object for a resource are applied via the @Redfish.SettingsApplyTime annotation.</td>
</tr>
<tr>
<td>1.2.1</td>
<td>2017-08-10</td>
<td>Clarified wording of the Oem object definition.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Clarified wording of the Partial resource results clause.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Clarified behavior of a service when receiving a PATCH with an empty JSON object.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Added statement about other uses of the HTTP 503 Service Unavailable status code.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Clarified format of URI fragments to conform to RFC6901.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Clarified use of absolute and relative URIs.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Clarified definition of the target property as originating from OData.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Clarified distinction between hyperlinks and the links property.</td>
</tr>
<tr>
<td>Version</td>
<td>Date</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Corrected the JSON example of the privilege map.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Clarified format of the @odata.context property.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Added clauses about the schema file naming conventions.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Clarified behavior of a service when receiving a PUT with missing properties.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Clarified valid values in the Accept header to include wildcards per RFC7231.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Corrected ConfigureUser privilege to be spelled ConfigureUsers.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Corrected the Session login clause to include normative language.</td>
</tr>
<tr>
<td>1.2.0</td>
<td>2017-04-14</td>
<td>Added support for the Redfish Composability Service.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Clarified service handling of the Accept-Encoding header in a request.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Improved consistency and formatting of example requests and responses throughout.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Corrected usage of the @odata.type property in response examples.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Clarified usage of the required annotation.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Clarified usage of Subordinate Overrides in the Privilege Registry.</td>
</tr>
<tr>
<td>1.1.0</td>
<td>2016-12-09</td>
<td>Added Redfish Service operation-to-privilege mapping clause. This functionality enables a service to present a resource or even property-level mapping of HTTP operations to roles and privileges.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Added references to the Redfish Host Interface Specification (DSP0270).</td>
</tr>
<tr>
<td>1.0.5</td>
<td>2016-12-09</td>
<td>Errata release. Various typographical errors.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Corrected the use of collection, resource collection, and members throughout.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Added glossary entries for resource collection and members.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Corrected certificate requirements to reference definitions and requirements in RFC5280 and added a normative reference to</td>
</tr>
<tr>
<td>Version</td>
<td>Date</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>------------</td>
<td>-------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>RFC5280.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Clarified usage of the HTTP POST and PATCH operations.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Clarified usage of the HTTP status codes and error responses.</td>
</tr>
<tr>
<td>1.0.4</td>
<td>2016-08-28</td>
<td>Errata release. Various typographical errors.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Added example of an HTTP Link Header and clarified usage and content.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Added the Schema modification clause, which describes the allowed usage of the schema files.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Added recommendation to use TLS 1.2 or later, and to follow the SNIA TLS Specification. Added reference to the SNIA TLS Specification. Added additional recommended TLS_RSA_WITH_AES_128_CBC_SHA cipher suite.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Clarified that the Id property of a Role resource matches the role name.</td>
</tr>
<tr>
<td>1.0.3</td>
<td>2016-06-17</td>
<td>Errata release. Fixed the missing numbering in the table of contents and clauses. Corrected URL references to external specifications. Added missing normative references. Corrected typographical error in ETag example.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Clarified examples for @Message.ExtendedInfo to show arrays of messages.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Clarified that a POST to Session Service to create a new session does not require authorization headers.</td>
</tr>
<tr>
<td>1.0.2</td>
<td>2016-03-31</td>
<td>Errata release. Various typographical errors.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Corrected normative language for M-SEARCH queries and responses.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Corrected Cache-Control and USN format in M-SEARCH responses.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Corrected schema namespace rules to conform to OData namespace requirements and updated examples throughout the document to conform to this format. Specifically, namespace.n.n.n becomes namespace.vn_n_n. File naming rules for JSON Schema and CSDL (XML) schemas were also corrected to match this format and to enable future major (v2) versions to coexist.</td>
</tr>
<tr>
<td>Version</td>
<td>Date</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>------</td>
<td>-------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Added clause that details the location of the schema repository and lists the durable URLs for the repository.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Added definition for the value of the Units annotation, using the definitions from the UCUM Specification. Updated examples throughout to use this standardized form.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Modified the naming requirements for Oem property naming to avoid future use of colon : and period . in property names, which can produce invalid or problematic variable names when used in some programming languages or environments. Both separators have been replaced with underscore (_), with colon (:) and period (.) usage now deprecated (but valid).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Removed duplicative or out-of-scope sub-clauses from the Security clause, which made unintended requirements on Redfish Service implementations.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Added the requirement that property names in resource responses match the casing (capitalization) as specified in schema.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Updated normative references to current HTTP RFCs and added clause references throughout the document where applicable.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Clarified ETag header requirements.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Clarified that no authentication is required for accessing the Service Root.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Clarified description of retrieving resource collections.</td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Clarified usage of</strong> charset=utf-8 <strong>in the HTTP Accept and Content-Type headers.</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Clarified usage of the Allow HTTP response header and added a table entry for the Retry-After header usage.</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td><strong>Clarified normative usage of the type property and context property, explaining the ability to use two URL forms, and corrected the @odata.context URL examples throughout.</strong></td>
</tr>
<tr>
<td></td>
<td></td>
<td>Corrected inconsistent terminology throughout the resource collection response clause.</td>
</tr>
<tr>
<td>Version</td>
<td>Date</td>
<td>Description</td>
</tr>
<tr>
<td>---------</td>
<td>------------</td>
<td>---------------------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Corrected name of normative resourceMembers property (Members, not value).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Clarified that error responses may include information about multiple error conditions.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Corrected name of Measures.Unit annotation term as used in examples.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Corrected outdated reference to Core OData Specification in annotation term examples.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Added the Members property to the Common Redfish resource properties clause.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Clarified terminology and usage of the task monitor and related operations in the Asynchronous operations clause.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Clarified that implementation of the SSDP protocol is optional.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Corrected typographical error in the SSDP USN field's string definition (now ::dmtf-org).</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Added the OPTIONS method to the allowed HTTP methods list.</td>
</tr>
<tr>
<td>1.0.1</td>
<td>2015-09-17</td>
<td>Fixed nullablity in example.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Clarified normative use of long description in schema files.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Clarified usage of the rel-describedby Link header.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Corrected text in example of &quot;Select List&quot; in OData context property.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Clarified Accept-Encoding request header handling.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Deleted duplicative and conflicting statement on returning extended error resources.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Clarified relative URI resolution rules.</td>
</tr>
<tr>
<td>1.0.0</td>
<td>2015-08-04</td>
<td>Clarified USN format.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Initial release.</td>
</tr>
</tbody>
</table>